Bachelor Thesis
Im Studiengang »Medien und Informationswesen«

Integration des Mikrocontrollers Arduino in vernetzte Umgebungen am Beispiel einer Monitor-Anwendung im Gesundheitswesen

vorgelegt von:
Florian Lutz

Erstbetreuer:
Prof. Dr. Tom Rüdebusch

Zweitbetreuer:
Prof. Dr. Volker Sänger

Hochschule Offenburg
Medien und Informationswesen
Wintersemester 2013/2014

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Berlin, den _____

Florian Lutz

INHALTSVERZEICHNIS

Inhaltsverzeichnis ..... **I**

Abbildungsverzeichnis ..... **V**

Tabellenverzeichnis ..... **IX**

1 Einführung **11**
1.1 Physical Computing 11
1.2 Projektumfeld 11
1.2.1 Arduino-Plattform 12
1.2.2 Arduino-Software 13
1.2.3 Programmiersprache 14
1.2.4 Arduino-Hardware 14
1.3 Alternative Boards 15

2 Recherche: Grundlagen und Analyse des Technologiestands **19**
2.1 Arduino-Projekte 19
2.1.1 Twitter-/Google/App-Projekte 20
2.1.2 Hausautomation 27
2.2 Arduino Funktionalität 32
2.2.1 Grundfunktionen 33
2.2.2 Atmel-Mikrocontroller 33
2.2.3 Programmierschnittstellen 35
2.2.4 General I/O 36
2.2.5 Stromversorgung 36
2.3 Sensorik 37
2.3.1 Bewegungs- und Beschleunigungs-Sensor 37
2.3.2 Licht-Sensor 38
2.3.3 Ultraschall-Sensor 38
2.3.4 Vibrations-Sensor 38
2.3.5 Temperatur-Sensor 38
2.3.6 Rotationswinkel-Sensor 39
2.3.7 E-Health-Sensoren 39
2.4 Bus-Systeme 39
2.4.1 UART und USART 39
2.4.2 1-Wire 40
2.4.3 I2C und SPI 41
2.5 Datenverarbeitung 43
2.5.1 EVA- und EVA(S)-Prinzip 43
2.5.2 Struktur eines Arduino-Sketches 44
2.5.3 Variablen, Datentypen und Konstanten 46
2.5.4 Testen und Debuggen 47
2.6 Bibliotheken 48
2.6.1 Standard-Bibliotheken 49
2.6.2 Bibliotheken von Drittanbietern 50
2.6.3 Bibliotheken manipulieren 50
2.6.4 Verwendung der Ethernet-Bibliothek 51
2.7 Hardwareerweiterungen (Shields) 53
2.7.1 Drahtloskommunikation 54
2.7.2 Netzwerkkommunikation 57
2.8 Arduino in Netzwerken 60
2.8.1 Grundlegende Begriffserklärung 60

2.8.2 Netzwerkkommunikation ... 61
2.8.3 Ethernet-Shield einrichten ... 63
2.8.4 Arduino als Web-Client ... 67
2.8.5 XML/JSON-Daten abrufen ... 71
2.8.6 Arduino als Web-Server ... 73
2.8.7 E-Mail-Versand mit Arduino ... 78
2.8.8 Arduino nutzt Twitter ... 80
2.8.9 UDP statt TCP ... 82
2.8.10 Cloud-Service »Xively« zur Daten-Visualisierung ... 82
2.9 Lokales Datenlogging und Visualisierung ... 87
2.9.1 Logging auf SD-Karten ... 87
2.9.2 Direkte Kommunikation mit dem Computer ... 88
2.10 Monitoring im Gesundheitswesen ... 89
2.10.1 Tonisch-klonischer Anfall ... 89
2.10.2 Systeme zur Überwachung ... 90
2.10.3 Richtlinie 93/42/EWG über Medizinprodukte ... 91
2.10.4 ZigBee Standard ... 92

3 Anforderungsanalyse ... 95
3.1 Projekt-Scope ... 95
3.2 Funktionale Anforderungserhebung ... 95
3.2.1 Eingabe/Erfassung ... 96
3.2.2 Verarbeitung ... 96
3.2.3 Ausgabe ... 97
3.2.4 Speicherung ... 98
3.2.5 Systeminteraktion ... 98
3.2.6 Funktionale Vereinbarung ... 98
3.3 Nicht-Funktionale Anforderungserhebung ... 98
3.3.1 Technische Anforderungen ... 99
3.3.2 Ergonomie ... 99
3.3.3 Bedienbarkeit ... 99
3.3.4 Leistung ... 99
3.3.5 Messgenauigkeit ... 100
3.3.6 Energieversorgung ... 100
3.3.7 Zuverlässigkeit ... 100
3.3.8 Zulassung gemäß Richtlinie 93/42/EWG ... 101
3.4 Analyse und Priorisierung der Anforderungen ... 101
3.4.1 Eingabe/Erfassung ... 101
3.4.2 Verarbeitung ... 101
3.4.3 Ausgabe ... 102
3.4.4 Speicherung ... 103
3.4.5 Systeminteraktion ... 103
3.4.6 Funktionale Vereinbarung ... 104
3.4.7 Technische Anforderungen ... 104
3.4.8 Ergonomie der Daten ... 105
3.4.9 Ergonomische Eigenschaften ... 105
3.4.10 Bedienbarkeit ... 105
3.4.11 Leistung ... 106
3.4.12 Messgenauigkeit ... 106
3.4.13 Energieversorgung ... 107
3.4.14 Zuverlässigkeit ... 107
3.4.15 Zulassung gemäß Richtlinie 93/42/EWG ... 107
3.5 Use-Case Diagramm ... 109
3.6 Zustandsautomat ... 110
3.7 Zusammenfassung ... 111
3.7.1 Anforderungen an den Prototyp ... 111

4 Konzept ... 113
4.1 Systemarchitektur und Komponenten ... 113
4.1.1 Architektur ... 113
4.1.2 Hardware-Komponenten ... 114
4.2 Server - Speichern und Visualisieren der Daten ... 115
4.3 Aktivitäten der Systeme ... 116
4.3.1 Arduino-Produzent ... 116
4.3.2 Arduino-Client ... 116
4.4 Projektumfang der Thesis ... 116

5 Implementierung ... 119
5.1 Überblick ... 119
5.1.1 Software ... 119
5.1.2 Architektur und Module ... 120
5.1.3 Testgetriebene Entwicklung und Debugging ... 121
5.1.4 Aufbau ... 122
5.2 Phase 1: Umsetzung Arduino-Produzent ... 122
5.2.1 Konfiguration der XBee-Module ... 123
5.2.2 Testen des Accelerometer ... 125
5.2.3 Testen des Puls-Oxigen-Sensors ... 132
5.2.4 Zusammenführen der Module ... 136
5.3 Phase 2: Einrichten von Xively ... 142
5.3.1 Konfiguration des Xively-Geräts ... 143
5.4 Phase 3: Umsetzung Arduino-Client ... 144
5.4.1 Testen des WiFi-Shields ... 145
5.4.2 Hinzufügen des XBee-Moduls ... 148
5.4.3 Verarbeitung des Textstrings ... 149
5.4.4 Einbinden von Xively in Arduino-Client ... 153
5.5 Phase 4: Finaler Systemtest ... 158
5.5.1 Phase 1: Stabilitätstest ... 158
5.5.2 Phase 2: Fehlfunktionstest ... 161
5.5.3 Überprüfen der ermittelten Anforderungen ... 163

6 Fazit ... 165
6.1 Zusammenfassung ... 165
6.2 Bewertung ... 166
6.3 Ausblick ... 166

Literaturverzeichnis ... I

Quellcodeverzeichnis ... IX

Anhänge ... XI
Anhang A: Pulse-Width-Modulation ... XI
Anhang B: Standards-Hausautomation ... XII
Anhang C: Anschlussbelegung ATMEGA328 ... XIV
Anhang D: Datentypen ... XVI
Anhang E: Ethernet Web-Client DNS-Sketch ... XVIII
Anhang F: WiFi Web-Client Yahoo-Sketch ... XIX
Anhang G: Ethernet Web-Server AnalogPin-Sketch ... XX
Anhang H: Mail-Header und Textkörper ... XXI
Anhang I: Xively Request-/Response-Header ... XXII
Anhang J: Pinbelegung Arduino-UNO (Produzent) ... XXIII
Anhang K: Pinbelegung Arduino-UNO (Client) ... XXIV

ABBILDUNGSVERZEICHNIS

Abbildung 1: Arduino-Entwicklungsumgebung 1.0.5 ... 13
Abbildung 2: Development Cycle [11] ... 13
Abbildung 3: Der Compiler als Dolmetscher [2] ... 14
Abbildung 4: Kommunikationsarchitektur – Besucherzähler mit Google-Analytics ... 20
Abbildung 5: Google Analytics Darstellung [24] ... 21
Abbildung 6: Aufbau ohne Shield – Besucherzähler Google Analytics [24] ... 21
Abbildung 7: Kommunikationsarchitektur - Twitter Mood Light ... 23
Abbildung 8: The Mood Light [27] ... 24
Abbildung 9: Temperatur-Sensor-App Aufbau [30] ... 25
Abbildung 10: Kommunikationsarchitektur - Temperatur-Sensor über iOS-Applikation auslesen ... 26
Abbildung 11: Temperature-Sensor iOS Applikation [30] ... 27
Abbildung 12: Kommunikationsarchitektur – Android-Türöffner ... 29
Abbildung 13: System-Aufbau [31] ... 30
Abbildung 14: Bewässerungssystem [32] ... 31
Abbildung 15: Hardware-Bewässerungssystem [32] ... 32
Abbildung 16: Arduino-UNO R3 [35] ... 33
Abbildung 17: Blockdiagramm des ATmega328 [36] ... 34
Abbildung 18: Ein 1-Wire Master mit ein oder mehreren 1-Wire Slaves ... 40
Abbildung 19: Ein I2C-Master mit ein oder mehr I2C-Slaves [5] ... 41
Abbildung 20: Signalanschlüsse für SPI-Master und –Slaves [5] ... 42
Abbildung 21: EVA-Prinzip ... 43
Abbildung 22: EVA(S)-Prinzip [49] ... 44
Abbildung 23: Grundlegende Sketch-Struktur [2] ... 45
Abbildung 24: Arduino Wireless-SD-Shield [59] ... 55
Abbildung 25: Adafruit PN532 NFC/RFID-Controller-Schild [62] ... 56
Abbildung 26: BLE Shield v2.0 RedBearLab [66] ... 56
Abbildung 27: Sparkfun WiFly Shield [71] ... 58
Abbildung 28: Arduino Ethernet Shield [73] ... 58
Abbildung 29: Arduino Yun – Systemarchitektur [74] ... 59
Abbildung 30: Client/Server-Kommunikation ... 62
Abbildung 31: Arduino-UNO - Ethernet-Shield Schaltplan ... 63

Abbildung 32: DHCP-Ablauf 65
Abbildung 33: Domain-Name-Server 66
Abbildung 34: Yahoo-Sketch - Serieller Monitor Yahoo-Anfrage 71
Abbildung 35: Arduino Web-Server Kommunikationsarchitektur 73
Abbildung 36: Arduino-Server - Browser-Screenshot 75
Abbildung 37: Arduino-Mail - Screenshot Apple-Mail 79
Abbildung 38: Kommunikationsarchitektur, Arduino twittert über ThingSpeak-Proxy 81
Abbildung 39: Die Protokolle zwischen dir und dem Web [3] 82
Abbildung 40: Xively Daten-Hierarchie [96] 83
Abbildung 41: Arduino-Xively - Xively-Sensortest 85
Abbildung 42: Kommunikationsarchitektur - Arduino-Client/Xively-Server 85
Abbildung 43: Arduino-Xively - Sensortest Temperatur und Licht 86
Abbildung 44: Processing-Visualisierung [99] 89
Abbildung 45: ZigBee-Stack [108] 93
Abbildung 46: IEEE 802.15.4 Netztopologie 93
Abbildung 47: ZigBee erweiterte Netztopologie 94
Abbildung 48: Liniendiagramm Zeit/Wert-Verlauf [111] 97
Abbildung 49: Use-Case Diagramm 109
Abbildung 50: Zustandsautomat 110
Abbildung 51: Systemarchitektur - Arduino-Anwendung im Gesundheitswesen 114
Abbildung 52: Kommunikationsarchitektur des Prototypen 120
Abbildung 53: XBee Explorer mit zwei XBee S1 Modulen 123
Abbildung 54: CoolTerm Ausgabe bei XBee-Konfiguration 124
Abbildung 55: CoolTerm XBee überprüfen 125
Abbildung 56: Arduino mit Accelerometer 127
Abbildung 57: Ausgabe Serieller-Monitor, Accelerometer 129
Abbildung 58: Accelerometer Processing Test 130
Abbildung 59: Arduino-Uno, Sensor-Shield und Puls-Oxigen-Sensor 133
Abbildung 60: Ausgabe Serieller Monitor, Puls-Oxigen-Sensor 135
Abbildung 61: Systemaufbau Arduino-Produzent 137
Abbildung 62: Testausgabe Serieller-Monitor Arduino-Produzent 139
Abbildung 63: Systemtest Arduino-Produzent 141
Abbildung 64: Ergebnis des Systemtests - Arduino-Produzent 142
Abbildung 65: Xively-Workbench 144

Abbildung 66: Arduino-UNO und WiFi-Shield 146
Abbildung 67: Ausgabe WiFi-Status am Seriellen-Monitor 147
Abbildung 68: Administrationsbereich des Routers im Netzwerk: o2-WLAN57 148
Abbildung 69: Arduino-Client Hardware 149
Abbildung 70: Funktionstest des Parsers 153
Abbildung 71: Screenshot Xively HTTP-Request und Response 156
Abbildung 72: Screenshot Xively-Workbench mit finalem Prototyp 157
Abbildung 73: Xively Pulsmessung im Systemtest 159
Abbildung 74: Xively Sauerstoffsättigungsmessung im Systemtest 159
Abbildung 75: Xively Bewegungsmessung im Systemtest 160
Abbildung 76: Überlappung der Messungen im Systemtest 160
Abbildung 77: Fehlfunktionstest Xively 161
Abbildung 78: Pulse Width Modulation [26] XI
Abbildung 79: ATMEGA328 Belegung [115] XIV
Abbildung 80: Xively Request-Header XXII
Abbildung 81: Xively Response-Header XXII

TABELLENVERZEICHNIS

Tabelle 1: ATmega328P-PU [38] 35
Tabelle 2: SPI and I2C Comparison [34] 43
Tabelle 3: Werte-Überprüfung Puls-Oxigen-Sensor 136
Tabelle 4: Vergleichstabelle des Systemtests - Arduino Produzent 142
Tabelle 5: Kommunikationsstandards der Home Automation XIII
Tabelle 6: Pins zu Anschlüsse auf Arduino-UNO [8] XV
Tabelle 7: Wichtige Datentypen für Arduino-Programme [5] XVII
Tabelle 8: SMTP-Sitzung, Mail-Header und Textkörper [116] XXI
Tabelle 9: Pinbelegung Arduino-Produzent [117]–[119] XXIII
Tabelle 10: Pinbelegung Arduino-Client [69], [119] XXIV

1 Einführung

1.1 Physical Computing

Unter dem Begriff »Physical Computing« ist weitestgehend ein interaktives, physisches System durch die Verwendung von Soft- und Hardware zu verstehen. Diese Systeme stellen eine Beziehung zwischen Menschen und Computern her. Dabei ist der Mensch ein Objekt aus der realen, physischen Welt und gilt somit als analog und Computer sollen sinnbildlich für die virtuelle, die digitale Welt stehen. Auch wenn überwiegend in der Literatur die Sprache von einem Menschen ist, so können als Akteure ebenfalls alle in der realen Welt vorkommenden Lebewesen, Geräte, Objekte und vieles mehr als diese fungieren. Es handelt sich daher um eine Beziehung zwischen der analogen und der digitalen Welt. [1], [2]

> „When asked to draw a computer, most people will draw the same elements: screen, keyboard, and mouse. When we think "computer," this is the image that comes to mind. In order to fully explore the possibilities of computing, you have to get away from that stereotype of computers. You have to think about computing rather than computers." [3]

Im obigen Zitat erklären Dan O'Sullivan und Tom Igoe wie der Mensch meist mit dem Begriff Computer umgeht und sich nicht von diesem stereotypischen Bild des Home-Computers löst. Um in die Welt des Physical Computings allerdings einsteigen zu können, ist es von Nöten sich von eben diesem Stereotypen zu lösen und das Bewusstsein zu erweitern, um die Vielfalt und Möglichkeiten von Rechenmaschinen, Chips und Prozessoren erfassen zu können. Dahingehend ist Physical Computing eher als Methode des »Interaction Designs[1]« zu verstehen und beschreibt daher ausschließlich die Art und Weise der Interaktion. [3], [4]

Durch die Verwendung von Sensoren und Mikrocontrollern können Signale an eine Software übertragen werden, welche wiederum durch Algorithmen Daten auswertet und weiterverarbeitet und/oder elektromechanische Geräte steuert. Am häufigsten wird der Begriff für künstlerische und designbasierte Projekte verwendet. [1], [2], [5]

1.2 Projektumfeld

Zur Entwicklung von Prototypen im Bereich Physical Computing eignen sich integrierte Systeme, welche eine benutzerfreundliche Entwicklungsumgebung, Software und Hardware vereinen. Bei der Umsetzung eines Einzelstückes oder einer Kleinserie lassen sich kostengünstige Systeme finden und somit auch eine rentable Entwicklung gewährleisten. Trotz der großen Vielfalt an Systemen, gilt Arduino als eines der am meist verbreitetsten und flexibelsten dieser Gruppe. Daher konzentriert sich diese Thesis auf die Arduino-Plattform, dennoch werden in diesem Kapitel auch die alternativen Boards dargestellt, um einen Überblick über alternative Systeme zu schaffen. [6]

[1] zu deutsch »Interaktionsgestaltung«

1.2.1 Arduino-Plattform

Arduino ist erstmals 2005 erschienen, von Studenten und Dozenten des Interaction Design Institute Ivrea (IDII)[2] entwickelt und anschließend publiziert. Zum damaligen Zeitpunkt setzte sich das Institut mit der Problematik auseinander, dass keine einfach anwendbaren und preiswerten Mikrocontrollersysteme für Studenten aus dem Design- und Kunstbereich erhältlich waren. Aus diesem Grund entwickelten sie in Zusammenarbeit mit Elektronikingenieuren ein eigenes integriertes System. Somit war die erste Serie des Arduino geboren. [8]

Seitdem entwickelt sich Arduino kontinuierlich weiter und adaptiert neue Technologien für diese Boards. Aufgrund der Verfügbarkeit der Hardware und Software auf Basis von Open-Source findet Arduino auch immer mehr Einzug in Seminare und Vorlesungen von Hochschulen und Universitäten. Es entstehen große Communities und Workshops rund um den Globus und zeigen somit die Faszination, die Einfachheit und Beliebtheit dieses Systems. Arduino erfreut sich zudem größter Beliebtheit bei Designern und Künstlern, welche mit diesen Boards auf einfache Art und Weise Prototypen entwickeln können, da aufgrund vieler Codebeispiele nur wenig technisches Verständnis von Nöten ist, aber dennoch sehr umfangreiche Projekte entwickelt werden können. Auch wenn Arduino benutzerfreundlich ist, bedeutet das nicht, dass es den von Ingenieuren entwickelten eingebetteten Systemen in irgendeiner Weise nachstehen müsse. [5], [9]

Der Begriff »Arduino« wird meist nur mit der Hardwarekomponente in Verbindung gebracht, umschließt allerdings auch den gleichnamigen Softwarebereich. Nur durch beide Komponenten kann die physikalische Welt gesteuert und wahrgenommen werden.

Open-Source

Software oder Lizenzen, welche mit diesem Begriff ausgezeichnet sind, werden als quelloffen bezeichnet. Damit benennt man einen öffentlich zugänglichen Quelltext, welcher je nach Lizenz meist frei kopiert, modifiziert und verändert werden kann. Die Open-Source Initiative (OSI) bewertet Software und überprüft diese auf Gültigkeit in Bezug auf die Open-Source Definition[3]. [10]

Wie bereits erwähnt, ist Arduino unter dem Begriff »Open-Source« einzuordnen. Durch die kostenlose Bereitstellung der Entwicklungsumgebung, wie auch die Möglichkeit der eigenen Herstellung der Hardware auf Basis von frei zugänglichen Schaltplänen, hat sich die Arduino-Community rasant vergrößert. Dabei stellen viele Entwickler ihre Projekte ins Internet und wiederum andere nutzen diese und entwickeln sie weiter. Des Weiteren werden nicht nur Programmcodes neu beziehungsweise weiterentwickelt, sondern auch die Boards, wodurch immer wieder neue Systeme entstehen, die es ermöglichen den aktuellen Stand verschiedenster Technologien in ein Arduino-Projekt zu implementieren. [2], [8], [9]

---

[2] Dieses Institut war eine unabhängige Non-Profit Organisation gegründet von Telecom Italia und Olivetti, ein Unternehmen für Computer- und Bürogeräte wie auch Anwendungssoftware. Es beinhaltete ein zweijähriges Masterprogramm mit Fokus auf Design, Architektur, Computerwissenschaften und Psychologie. [7]

[3] siehe *http://de.wikipedia.org/wiki/Open_Source_Initiative#Definition_von_Open_Source*

1.2.2 Arduino-Software

Von Arduino wird eine Entwicklungsumgebung zur Verfügung gestellt, die ebenfalls den Namen »Arduino« trägt. Durch diese Umgebung ist es möglich, Programme, sogenannte »Sketches«, auf dem Computer zu schreiben und anschließend durch den in der Entwicklungsumgebung implementieren Compiler in Instruktionen zu wandeln, welche das Arduino-Board beziehungsweise der Mikrocontroller interpretieren kann. Der Sketch wird über einen USB-Port per serieller Datenübertragung von der Entwicklungsumgebung in den Speicher des Boards geladen. Während der Entwicklungs- und Testphase muss diese physische Verbindung aufrecht erhalten bleiben. [5], [8]

Abbildung 1 zeigt ein Screenshot der typischen Arduino-Software. Diese ist so simpel wie möglich gehalten, damit auch hier keine Probleme oder Missverständnisse auftreten können.

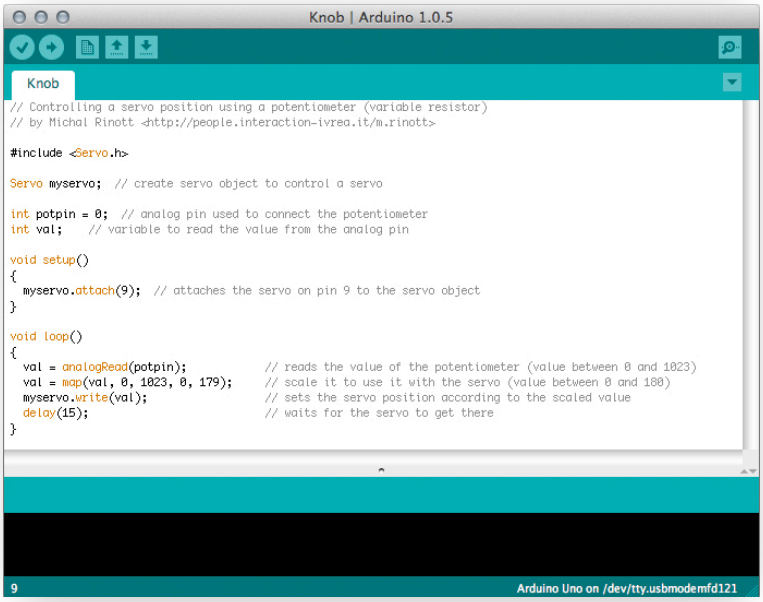


Abbildung 1: Arduino-Entwicklungsumgebung 1.0.5

Betrachtet man den Entwicklungszyklus eines Arduinos, so fällt auf, dass anders zur herkömmlichen Entwicklung für den Web-Bereich statt Editierung und Ausführung noch zwei weiter Phasen benötigen werden, um den Programmcode erfolgreich auf das Board zu laden.

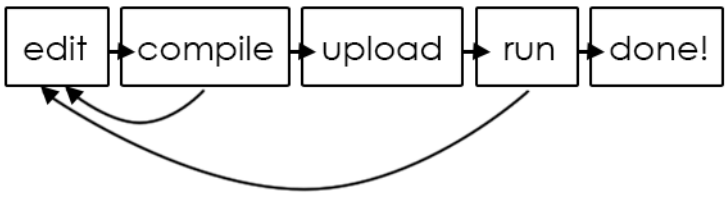


Abbildung 2: Development Cycle [11]

In Abbildung 2 ist zu erkennen, dass in zwei Phasen im Entwicklungszyklus der Code editierbar ist. Zum einen sollte die Kompilierung aufgrund von Fehlern im Programmcode oder nicht ausreichender Speicherkapazität fehlschlagen und zum anderen kann auch bei der Ausführung des Sketches der Code weiter editiert werden, da zu diesem Zeitpunkt das Programm schon auf das Arduino-Board überspielt wurde und alle Änderungen ab diesem Zeitpunkt nach dem Upload erst durch ein erneutes Hochladen überspielt werden. Arduino ist ein geschlossenes, autarkes System, welches selbstständig arbeitet und lediglich mit Strom versorgt werden muss. Dahingehend kann der auf dem Board befindliche Programmcode nicht verändert oder manipuliert werden, außer wenn Programmcode neu auf das System geladen wird.

1.2.3 Programmiersprache

Die Arduino-Programmiersprache basiert auf »Wiring[4]«, eine Open-Source Programmierumgebung für Mikrocontroller. Der Programmcode wird in einer Art eines vereinfachten Dialekts von »C/C++« geschrieben, welcher ebenfalls den Namen »Arduino« trägt. Erweitert werden kann die Sprache durch C++-Bibliotheken und des Weiteren ist es möglich direkt in »AVR C« zu programmieren. Oftmals bringt man die Arbeit mit Mikrocontrollern und das Ansteuern von Hardware, mit kryptischem Code in »Assembler« in Verbindung, was meistens auch zutrifft, doch das wurde bei der Entwicklung bewusst gemieden. Arduino als Programmiersprache wurde weitestgehend so entwickelt, dass keine tieferen Programmierkenntnisse von Nöten sind, um mit Arduino erste kleine Projekte zu realisieren. Der Mikrocontroller versteht natürlich die Sprache C/C++ nicht, daher wird der Programmcode, wie in folgender Abbildung dargestellt, über einen Compiler in Maschinensprache übersetzt. [2], [8], [12], [13]

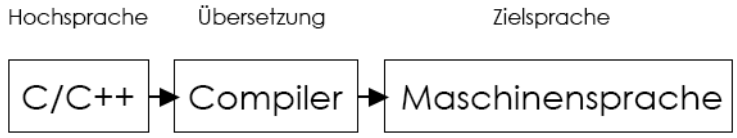


Abbildung 3: Der Compiler als Dolmetscher [2]

Da die meisten Boards über AVR-Hardware verfügen, ist es ebenfalls möglich diese Hardware in Assembler, C und vielem mehr zu programmieren. Arduino als Programmiersprache soll lediglich die Programmierung vereinfachen. [14]

1.2.4 Arduino-Hardware

Ist ein Sketch auf das Board geladen, so wird der geschriebene Code auf diesem ausgeführt. Da die Hardware ausschließlich auf Strom reagieren beziehungsweise diesen steuern kann, befinden sich auf dem Board Ein- und Ausgänge, worüber Sensoren und Aktuatoren angeschlossen, gesteuert und überwacht werden können. Sensoren sind dafür zuständig, Werte und bestimmte Aspekte aus der realen, physischen Welt in Strom umzuwandeln, worauf die Arduino-Hardware anschließend reagieren kann. Aktuatoren hingegen erhalten Strom vom Board und wandeln diesen zum Beispiel in eine mechanische Bewegung um. [5]

[4] ebenfalls eine Physical-Computing-Plattform, ähnlich wie Arduino

Arduino-Board

Unter dem Begriff »Boards« werden mit Komponenten bestückte Leiterplatten verstanden. Diese elektronischen Komponenten sind unter anderem Schaltkreise, Widerstände, Kondensatoren und vieles mehr. Zusätzlich sind diese Leiterplatten mit verschiedenen Anschlüssen und Schnittstellen versehen, um eine Verbindung in die physische Welt herzustellen. Das Standardboard von Arduino trägt die Bezeichnung »Arduino-UNO« (Stand 2012) und die aktuelle Version trägt den Zusatz »Rev 3«. Diese Boards können von verschiedenen Anbietern bezogen werden, einige dieser bieten Einsteiger-Kits an, bestehend aus Widerständen, Dioden, Sensoren, Aktuatoren und vielem mehr, um den Einstieg zu erleichtern und den direkten Start durch einfache Projekte zu realisieren. Hierbei wird meist der Arduino-UNO mitgeliefert, welcher sich zum Einstieg und für erste Projekte sehr gut eignet. [8]

Es gibt eine Vielzahl an verschiedenen Boards für zahlreiche Anwendungsfälle. Zurzeit bietet Arduino auf der Webseite 23 verschiedene Boards an, darunter welche zum Einnähen in Textilien und andere für den Robotik-Bereich. [15]

Durch das Open-Source-Prinzip entstehen auch sogenannte kompatible Klone für weitere Anwendungsfälle. Des Weiteren sind auch sogenannte Shields erhältlich, um einem Arduino neue Anwendungsmöglichkeiten zur Verfügung zu stellen. Diese können zum Beispiel einem Arduino-UNO, welcher ohne WiFi-Funktionalität ausgestattet ist, um eben diese erweitern. Diese Shields können wiederum direkt von Arduino, wie auch von Drittanbietern, bezogen werden. Auf Shields und Boards wird in Kapitel 2 genauer eingegangen. [9], [15]

1.3 Alternative Boards

Ein Board zum Entwickeln eines Prototyps muss nicht von Arduino bezogen werden. Wie im vorhergegangenen Kapitel angesprochen, existieren aufgrund der Open-Source-Hardware etliche Klone auf dem Markt. Dabei handelt es sich nicht um schlechte Kopien eines Arduino-Boards. Sie sind entweder für einen speziellen Anwendungsfall entwickelt oder eine kostengünstigere Variante zu Arduino.

Allerdings sind nicht alle Alternativen kompatibel mit der Arduino Soft- und Hardware. Dahingehend muss die Auflistung kategorisiert werden in soft- und hardwarekompatible Alternativen zu Arduino, erweiterungsimplementierte, alleinig softwarekompatible, nicht ATmega und Non-Arduino-Boards. Um einen Überblick zu erlangen, folgt nun eine reduzierte Auflistung der momentan verfügbaren Boards. Einige dieser tragen einen sehr ähnlichen Bezeichner. Das liegt daran, dass der Name »Arduino« geschützt ist und um dennoch eine Anlehnung an Arduino aufzuzeigen, wählen einige Hersteller die Endung »–duino«. [16]

Soft- und hardwarekompatible Boards

Diese Boards sind teilweise, meist aber vollständig kompatibel zu der Arduino-Soft- und Hardware.

- **SaintSmart UNO R3 ATmega328-AU**
  Dieses Board ist voll kompatibel zum Arduino-UNO R3, liegt preislich aber bei nur zirka 14 Euro, wohingegen der Arduino-UNO R3 zirka 8 Euro teurer ist. [17]
- **RoMeo-All in one Controller**
  Dieser ist ebenfalls voll kompatibel zu Arduino und wurde für den Robotik-Bereich konzipiert und verfügt über einen 2-Wege DC Motortreiber und Wireless-Socket. Bei diesem Board ist daher kein weiteres Shield für Motorsteuerung und WiFi-Funktion nötig. [18]

Boards mit Erweiterungen

Dieser Typ von Board ergänzt eine Leiterplatine auf Basis des Arduinos um weitere Hardware für eine spezielle Anwendung, ähnlich dem Anwendungsfall eines Arduino-Boards mit zusätzlichem Shield. Allerdings sind nicht alle Alternativen kompatibel zu Arduino. [16]

- **Lightuino 5.0**
  Wie der Name schon verrät, handelt es sich hierbei um ein Board, welches auf die Steuerung von Lichtquellen spezialisiert ist. Mit dem Lightuino lassen sich bis zu 1120 LEDs ansteuern, zum Vergleich kann der Arduino-UNO R3 lediglich 64 LEDs ohne Erweiterungs-Shield ansteuern. [19]
- **JeeNode (v6)**
  Dieses Board ist ein Arduino-kompatibles Funksignal-Board, welches über ein dementsprechendes RFM12B-Funkmodul verfügt. [20]

Ausschließlich softwarekompatible Boards

Folgende Boards schließen eine Hardwarekompatibilität aus, das kann unter anderem durch nicht akzeptierte Standardshields, andere Anschlüsse für die Stromversorgung wie auch der Ein- und Ausgänge entstehen. Bei diesen Boards steht meist der Kostenfaktor im Vordergrund weshalb oftmals auf einen USB-Anschluss verzichtet wird. [16]

- **Bare Bones Board (BBB)**
  Dieses Board ist eine kostengünstige und kompakte wie auch kompatible Alternative zu Arduino für Versuchsaufbaue. Meist nur als Kit, dafür sehr preiswert erhältlich. [21]

Boards ohne ATmega-Prozessor

Der Mikrocontroller bei diesen Boards ist nicht mit der Arduino-Entwicklungsumgebung kompatibel, allerdings besitzen einige dieser Boards eine eigene, aber ähnliche Entwicklungsumgebung, welche wiederum Arduino-Bibliotheken beinhalten. [16]

Non-Arduino-Boards

Diese Boards verfügen ebenfalls über keinen ATmega-Mikrocontroller und sind aufgrund dessen auch nicht zur Arduino-Entwicklungsumgebung kompatibel. Des Weiteren gibt es auch keine Möglichkeit die Arduino-IDE oder Software-Bibliotheken zu implementieren, mit diesen eine Kompatibilität hergestellt werden könnte. [16]

Raspberry Pi

Oftmals wird dieses Board in dieselbe Kategorie wie ein Arduino gesteckt, auch scheinen viele Anwendungsfälle dieselben zu sein, dennoch sind diese beiden Systeme nicht direkt vergleichbar.

Der Arduino wurde konzipiert, um von Grafikern, Designern oder Hobbybastlern genutzt zu werden. Dabei liegt das Augenmerk auf der Interaktivität, welche der Arduino ermöglicht. Des Weiteren kommt beim Arduino ein Mikrocontroller zum Einsatz, der auch alleinstehend funktionsfähig ist.

Das Raspberry Pi hingegen wurde entwickelt, um Schülern und Studenten das Programmieren beizubringen und sie in die Informatikwelt einzuführen. Daher wird dieses Board auch als kleiner Computer bezeichnet, da es einen Anwendungsprozessor besitzt und an einen Bildschirm angeschlossen werden kann und somit als normaler Desktop fungiert. Bei der Inbetriebnahme kann man verschiedene Installationsvarianten für das Betriebssystem wählen. Wird »NOOBS[5]« zur Hilfe genommen, stehen dem Benutzer momentan sechs Betriebssysteme zur Verfügung. [22], [23]

Allerdings haben viele Entwickler auch das Raspberry Pi für sich entdeckt und nutzen die preiswerte Variante, um Physical-Computing-Projekte umzusetzen oder verbinden dieses mit einem Arduino, um das daraus resultierende Leistungsspektrum zu erhalten. Daher ist Raspberry Pi keine Alternative zu Arduino, sondern vielmehr eine Erweiterung, mit der sich gerade im Bereich der Robotik einige große und rechenintensive Projekte komfortabel umsetzen lassen. [22], [23]

[5] NOOBS (New Out Of Box Software) ist ein bootfähiges Image für das Raspberry Pi, über das sich verschiedene Betriebssysteme installieren lassen [22]

2 Recherche: Grundlagen und Analyse des Technologiestands

Der Begriff »Do-It-Yourself (DIY)[6]« bildet den Grundgedanken, der hinter Arduino-Anwendungen steht. Benutzer sollen sich mit Physical Computing auseinandersetzen und die Möglichkeit erlangen, eigene Anwendungen zu entwickeln und zu programmieren. Vom professionellen Programmierer über Elektroingenieure bis hin zu Hobbybastlern erstreckt sich der Anwenderumfang derer, die Arduino für sich entdeckt haben. Dabei steht nicht im Vordergrund, sich im Voraus auf eine konkrete Anwendung mit detailliertem Schaltplan und Strukturgrammen zu fixieren als vielmehr um die Möglichkeit auszuprobieren, zu testen und Projekte wachsen zu lassen, assistierende Technologien zu verwenden und neue Erfahrungen und Ergebnisse mit Communities zu teilen. Aufgrund der Open-Source-Entwicklungsumgebung lassen sich zahlreiche Bibliotheken implementieren, verschiedene Shields zur Funktionalitätserweiterung anbringen und Hacks[7] auf andere Hardware durchführen.

Das Recherchekapitel befasst sich mit existierenden Projekten, spezifischere Einblicke in die Datenverarbeitung und Kommunikationsvarianten zwischen der Arduino-Hardware und anderen Geräten. Zudem wird aufgezeigt, welche Art der Verbindung zwischen Arduino und einem Gerät für einen bestimmten Anwendungsfall die geeignete ist. Dieses Kapitel dient als Grundlage für eine Anforderungsanalyse und Konzeption eines Prototyps, der über einen der recherchierten Kommunikationsvarianten verfügt.

2.1 Arduino-Projekte

Die Anwendungsbereiche integrierter Systeme könnten nicht vielseitiger als bei Arduino sein. Täglich realisiert die Arduino-Community interessante und nützliche Lösungen und publiziert diese auf verschieden Plattformen. Elektronikprojekte auf Basis eines Arduinos bestehen meist aus dem Arduino-Board und externen Komponenten, wie Widerständen, Leuchtdioden und Ansteuer- und Umformschaltungen für Sensoren und Aktuatoren. [8]

Dieses Kapitel soll einen groben Überblick über Arduino-Projekte schaffen und aufzeigen, wie man Arduino nutzen und zielorientiert einsetzen kann. Diese sollen ebenfalls als Anregung dienen und Anwendungsmöglichkeiten erschließen. Es folgen zwei Projektumfelder gegliedert in Twitter/Google/iOS Applikation und Hausautomation. Einige der Projekte sind ebenfalls in das jeweils andere genannte Projektumfeld einzuordnen, werden allerdings dem Hauptanwendungsfall zugeordnet. Diese zwei Bereiche umfassen nicht das gesamte Leistungsspektrum der zu realisierender Projekte, bilden allerdings einen kleinen Einblick in die Integration von Arduino in vernetzte Umgebungen.

[6] Der Begriff »Do-It-Yourself« kommt aus dem Englischen und bedeutet soviel wie »Mach es selbst«. In Bezug auf eine Entwicklung trägt es die Bedeutung »Marke Eigenbau«.

[7] Schaffen einer kreativen Lösung für Hardware- oder Programmierungsprobleme durch die Umgehung von hard- und softwareseitigen Einschränkungen.

2.1.1 Twitter-/Google/App-Projekte

In diesem Umfeld wird Arduino als Web-Server oder -Client eingesetzt und kommuniziert über einen Router. Die Kommunikationsmethoden, welche bei diesen Projekten zum Einsatz kommen, sind auch im folgenden Kapitel »Hausautomation« beliebte Varianten zur Kommunikation. Dennoch gibt es einige Projekte, welche nicht einer der genannten Anwendungsfälle angehören und dennoch diese Kommunikationsvariante zwischen Arduino und Netzwerken verwenden.

Besucherzähler mit Google Analytics[8]

Das Projekt namens »Log real life events in Google Analytics« soll die Möglichkeit der Visualisierung und Auswertung von Daten über Google Analytics erläutern. Als Beispiel soll ein Besucherzähler entwickelt werden, welcher alle Ankömmlinge zählt und in einem Diagramm in zeitlicher Abhängigkeit darstellt. Hierzu verwenden die Entwickler einen Drucksensor und Potenziometer, um die Sensitivität des Sensors zu regulieren (siehe Abbildung 6). Welches Arduino-Board verwendet wird, ist irrelevant, allerdings muss eine Konnektivität zum Router beziehungsweise ins Internet hergestellt werden können. Daher verwenden die Macher ein Ethernet-Shield und schließen dieses an den Router an. In diesem Fall muss Arduino als Web-Client fungieren und stellt ein HTTP-Request an Google Analytics (siehe Abbildung 4).

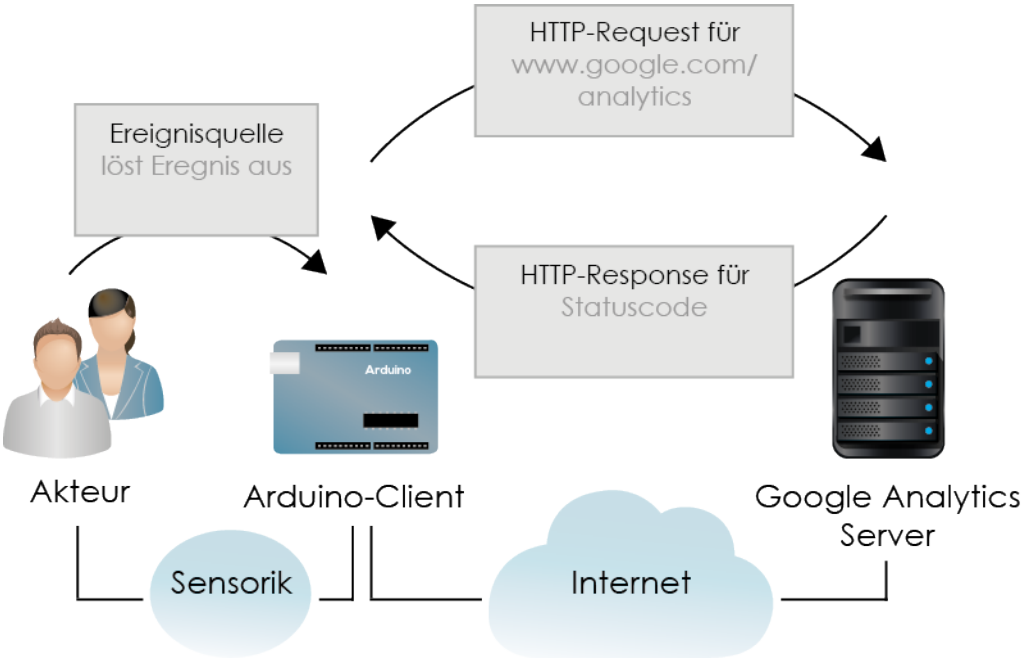


Abbildung 4: Kommunikationsarchitektur – Besucherzähler mit Google-Analytics

Schwieriger als beim HTTP-Request wird es beim Response, dieser muss manuell eingelesen und analysiert werden. Mit einer Schleife und einer C-Funktion (`sscanf`), welche den ankommenden Code untersucht, ist aber auch das umzusetzen. Dies dient ausschließlich zur Überprüfung, ob der Besucher gezählt wurde. Da der Response ausschließlich auf den HTTP-Statuscode überprüft wird, ist diese Funktion optional. Die Ausgabe erfolgt über Google Analytics (siehe Abbildung 5), womit die Entwickler zeigen

[8] Google Analytics ist ein kostenloser Dienst, welcher Zugriffe auf eine Website analysiert.

wollen, dass diese Plattform nicht ausschließlich für Webseiten oder mobile Applikationen dienen kann, sondern einen breiteren Anwendungsumfang umfasst, wie zum Beispiel Physical Computing-Projekte.

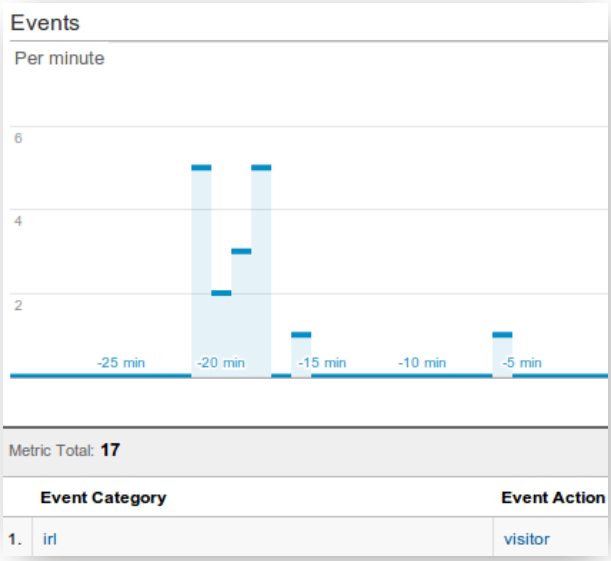


Abbildung 5: Google Analytics Darstellung [24]

Der Sensor muss anschließend unter den Fußabtreter gelegt und der Arduino mit Strom versorgt werden. Der Aufbau sieht wie folgt aus, allerdings ist bei der Darstellung von einem Shield abgesehen worden und die Leuchtdioden dienen ausschließlich als Kontrollleuchten in Bezug auf den HTTP-Response.

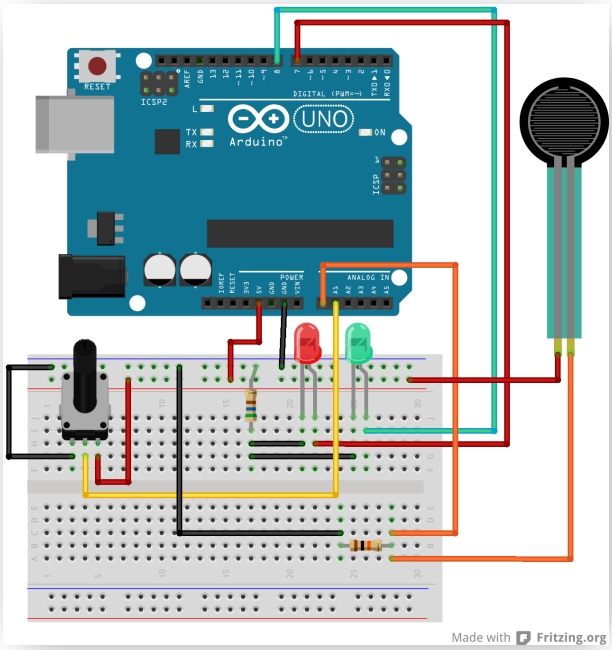


Abbildung 6: Aufbau ohne Shield – Besucherzähler Google Analytics [24]

Genauere Information wie auch der Programmcode ist unter *http://jolicode.com/blog/log-real-life-events-in-google-analytics* erhältlich. [24]

Twitter Mood Light – The World's Mood in a Box

Wie geht es den Menschen auf der Welt? Mit dieser Frage setzt sich dieses Projekt auseinander. Das »Twitter Mood Light« stellt über ein Farbspektrum den momentanen Gefühlzustand eines gewissen Durchschnittes an Twitter-Usern dar. Es soll sich an dem kollektiven Befinden der Menschen orientieren und wie der eigene Körper, welcher auf gewisse Art und Weise einem signalisiert wie das momentane Befinden ist, dem Anwender die Möglichkeit eröffnen nicht das eigene, sondern das kollektive Empfinden zu erfahren.

Die Hardware besteht im Wesentlichen aus einem Arduino-Board, einem WiFi-Shield (im Projekt das Sparkfun WiFly-Shield) und einer RGB Leuchtdiode. Folgende Bauteile verwendet der Entwickler von »Twitter Mood Light«.

- Arduino Duemilanove
- WiFly-Shield WRL-09954 von Sparkfun
- Break Away Headers – Straight von Sparkfun
- 9 V Batterie
- 9 V Barrel Jack Adapter
- 5 mm RGB LED
- drei Widerstände (zweimal 100 Ohm, einmal 180 Ohm)
- Draht
- eine kleine Leiterplatte
- USB-Kabel um Arduino mit Computer zu verbinden

Das Arduino-Board ist austauschbar, hierfür kann auch der neuere Arduino-UNO verwendet werden. Um die Netzwerkkommunikation zu gewährleisten, wird ein WiFly-Shield eingesetzt, welches das Arduino mit einem 802.11b/g WLAN-Netzwerk verbindet. Grundlage des Moduls bilden das WLAN-Modul und der SPI-zu-UART Controller. [25]

Zuerst wird das Arduino-Board mit dem WiFly-Shield verbunden und anschließend am Computer per USB-Kabel angeschlossen. Die RGB LED hat vier Anschlüsse, über welche die einzelnen Farben Rot, Grün und Blau angesteuert werden können. Der vierte Anschluss ist für Ground, die Masse. Das Arduino-Board verfügt über digitale und analoge Ein- und Ausgänge. Einige der digitalen Pins verwenden »PWM[9]«, und somit kann man auch an den digitalen Ausgängen 256 Werte steuern, von komplett AUS (0) bis vollständig AN (255), ähnlich wie bei einem Dimmer (siehe Anhang A). Dieser Wertebereich entspricht nicht dem der analogen Eingänge, bei denen dieser 1024 Werte beträgt. Ausgänge sind bei einem Arduino stets digital und werden durch PWM in analoge simuliert und per `analogWrite()`-Funktion beschrieben. An diesen Ausgängen werden die RGB-Anschlüsse angebracht, zusätzlich werden hierzu die Widerstände benötigt.

Das kollektive Befinden wird aufgrund von Tweets gemessen, dahingehend muss eine Verbindung zu Twitter aufgebaut werden. Eine der wichtigsten Aspekte bei diesem Projekt sind die Suchanfragen, um das Empfinden der Twitter-User herauszufinden. Twitter ermöglicht eine Suche nach den letzten Tweets[10], welche ein bestimmtes Wort oder ein Satzteil enthalten. Daher muss eine Liste mit dementsprechenden Satzteilen erstellt werden,

[9] PWM steh für Pulse-Width-Modulation mit dem Verfahren man an digitalen Ein- und Ausgängen ein analoges Signal simulieren kann. [26]

[10] Tweets sind die von Usern eingetragenen Nachrichten und Mitteilungen

nach denen die Tweets über eine »OR«-Funktion durchsucht werden. Diese Anfrage kann wie folgt aussehen.

> „GET/search.json?q="i'm+so+scared"+OR+"i'm+really+scared"+OR+"i'm+terrified"+OR+"i'm+really+afraid"+OR+"so+scared+i"&rpp=30&result_type=recent“ [27]

Twitter stellt in der »Search API[11]« eine JSON-Callback-Funktion zur Verfügung, welche hierbei verwendet wird.

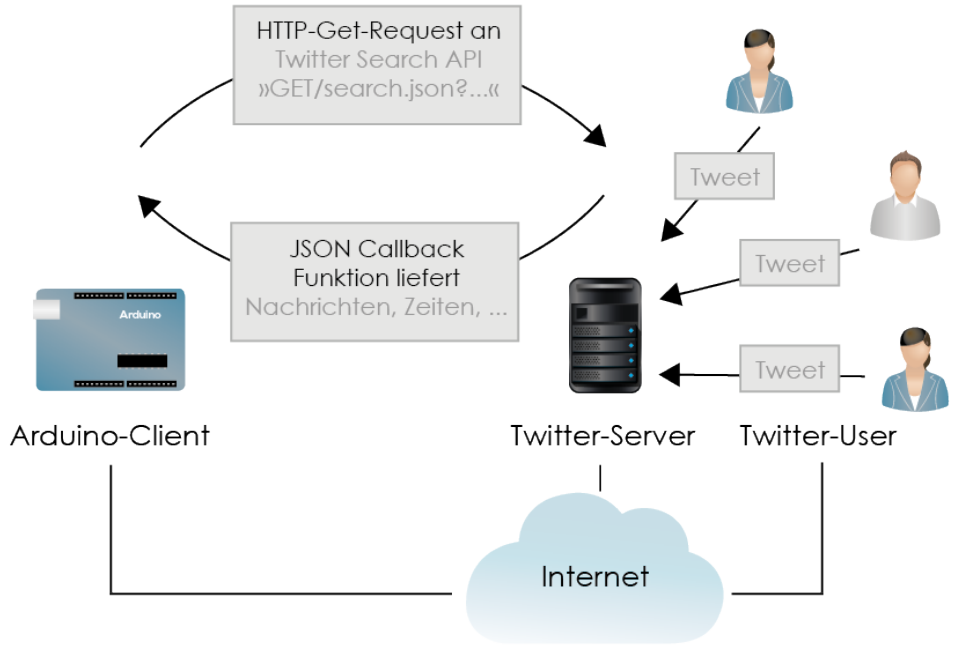


Abbildung 7: Kommunikationsarchitektur - Twitter Mood Light

Es hat sich herausgestellt, dass die Suche nach Adverbien und übertriebenen Gefühlszuständen wesentlich bessere und exaktere Ergebnisse liefert. Um emotionale Ausbrüche und daher nutzlose Tweets auszusortieren, verwendet der Entwickler eine Werteskala über den zeitlichen Gehalt eines Tweets. So werden kurzzeitige und somit irrelevante Tweets nicht in das Suchraster mit aufgenommen. Dadurch ist die Reaktionszeit verkürzt und der Algorithmus effektiver. Mit einer geeigneten Liste an Suchbegriffen ist auch diese Phase des Projekts abgeschlossen. Die Bibliotheken stellt der Entwickler auf der Projektseite zur Verfügung, daher muss nur das eigene WLAN-Setup vorgenommen und die Suchbegriffe mit den Eigenen ausgetauscht werden. Die Bibliotheken enthalten zudem die Zeit für den Übergang zwischen einer zu anderen Phase. Hierzu benötigt man die PWM-Anschlüsse, ohne die kein dimmender Übergang möglich ist. Das Resultat kann wie folgt aussehen.

[11] API ist eine Schnittstelle zur Anwendungsprogrammierung, wird von Softwaresystemen bereitgestellt um andere Programme an das System anbinden zu können

Abbildung 8: The Mood Light [27]

Genauere Information wie auch der Programmcode ist unter *http://www.instructables.com/id/Twitter-Mood-Light-The-Worlds-Mood-in-a-Box/?ALLSTEPS* einzusehen. [27]

Temperatur-Sensor mit iOS-Applikation

Die Intension des Entwicklers war dahingehend, dass die Applikationen, welche für iPhones zur Verfügung stehen, meist zu ungenau sind oder die Wetterstation, welche die Daten liefert, zu weit entfernt ist und somit falsche oder ungenaue Wetterdaten für den eigenen Standort übermittelt. Daher entwickelt dieser eine eigene Applikation und Arduino-Anwendung, mit der es möglich ist, verschiedene Temperatur-Sensoren auszulesen und die Ergebnisse per iOS-Applikation darzustellen. Folgende Hardwarekomponenten werden hierfür benötigt.

- Sparkfun Inventor's Kit for Arduino
- Arduino Ehternet-Shield
- Break Away Headers
- Temperatur-Sensoren
- Grove Base Adapter-Shield für Arduino

Das »Sparkfun Inventor's Kit« für Arduino ist nicht zwingend notwendig, die Komponenten sind auch einzeln und somit günstiger zu beziehen. Der »Grove Base Adapter« ist ein Shield mit einer großen Anzahl an Konnektoren. Diese erlauben es, alle Sensoren mit einem bestimmten Stecker zu versehen, um somit eine Verwechselung bei der Verdrahtung falscher Pins zu vermeiden. Daher ist es optional und kann durch das direkte Anschließen an dem Ethernet-Shield umgangen werden.

Die Temperaturproben verwenden eine »1-Wire-Bus[12]«-Technologie (siehe Kapitel 2.4.2), mit der es möglich ist aufgrund des One-Master/Multi-Slave-Prinzips, pro Bus zwar nur

[12] Dieses System, entwickelt von Dallas Semiconductor Corp., ist eine serielle Schnittstelle, welche als Stromversorgung wie auch Sende- und Empfangsleitung genutzt werden kann. [28]

einen Master (Steuereinheit) allerdings bis zu 100 Slaves (Sensoren oder Ähnliches) anzuschließen. Jeder Slave wird dabei mit einer 64-Bit-ROM[13]-ID adressiert. [29]

Dementsprechend wird nur ein Draht beziehungsweise eine Verbindung benötigt, um alle Geräte, welche am Bus angeschlossen sind, zu erreichen. Allerdings ist das für diesen Anwendungsfall nicht notwendig, da nur von zwei Sensoren ausgegangen wird. Sollten es allerdings zehn bis 20 Sensoren sein, dann müssen mehrere Sensoren auf einen digitalen Pin gesetzt werden. Auch wenn, wie in diesem Beispiel, pro Sensor ein eigener digitaler Pin verwendet werden kann, wird dennoch das 1-Wire-Kommunikationsprotokoll verwendet. Hierzu werden wiederum dementsprechende Bibliotheken angeboten, welche kostenfrei bezogen werden können. In einem späteren Kapitel wird auf die Implementierung von Bibliotheken noch eingegangen.

Ist das Arduino-Board mit dem Ethernet-Shield verbunden, wird dieses zusätzlich noch mit einer Leuchtdiode versehen, welche als Signal- und Statusleuchte fungiert. Wichtig ist, dass keine Pins belegt werden, welche von anderen Shields beim Aufbau benötigt werden.

Sinnvolle Pinbelegung bei diesem Projekt.

- Data-Pin 5: Temperatur-Sensor 1 (Indoor)
- Data-Pin 6: Temperatur-Sensor 2 (Outdoor)
- Data-Pin 7: Status und Signal Leuchtdiode
- Data-Pins 10-13: Ethernet Shield

Die Basis bildet das Arduino-Board, auf dem das Ethernet Shield angebracht ist und auf diesem wiederum das »Grove Base Adapter Shield« gesteckt ist. Die Pins, welche zur Verfügung stehen, sind dieselben wie auf dem Arduino-Board, da die Signale durchgeschleust werden.

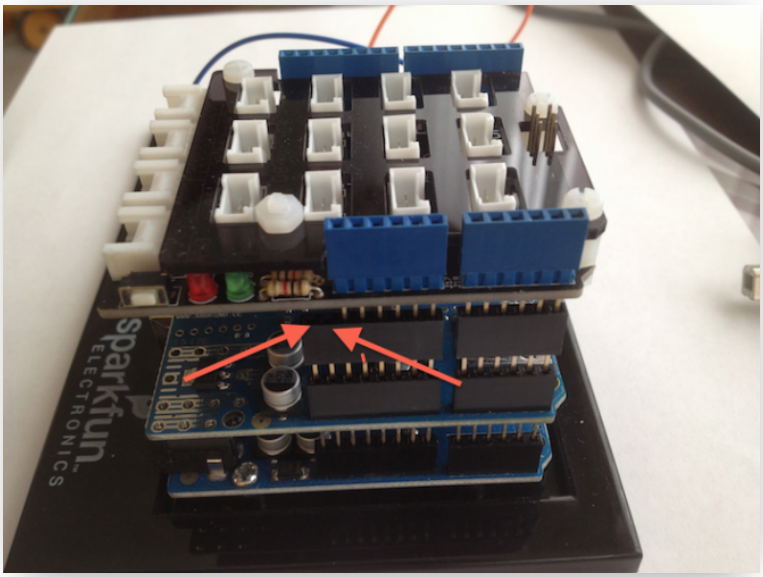


Abbildung 9: Temperatur-Sensor-App Aufbau [30]

[13] Akronym für »Read Only Memory«

Über die Ethernet-Bibliothek wird eine Verbindung zum Router aufgebaut und der Arduino ist anschließend fähig Daten zu übertragen und über eine generierte IP-Adresse lokal auf den Arduino zuzugreifen. Da der Arduino als Sever eingerichtet ist, kann dieser auf ein Request über die IP-Adresse, ein HTTP-Response zurückgeben.

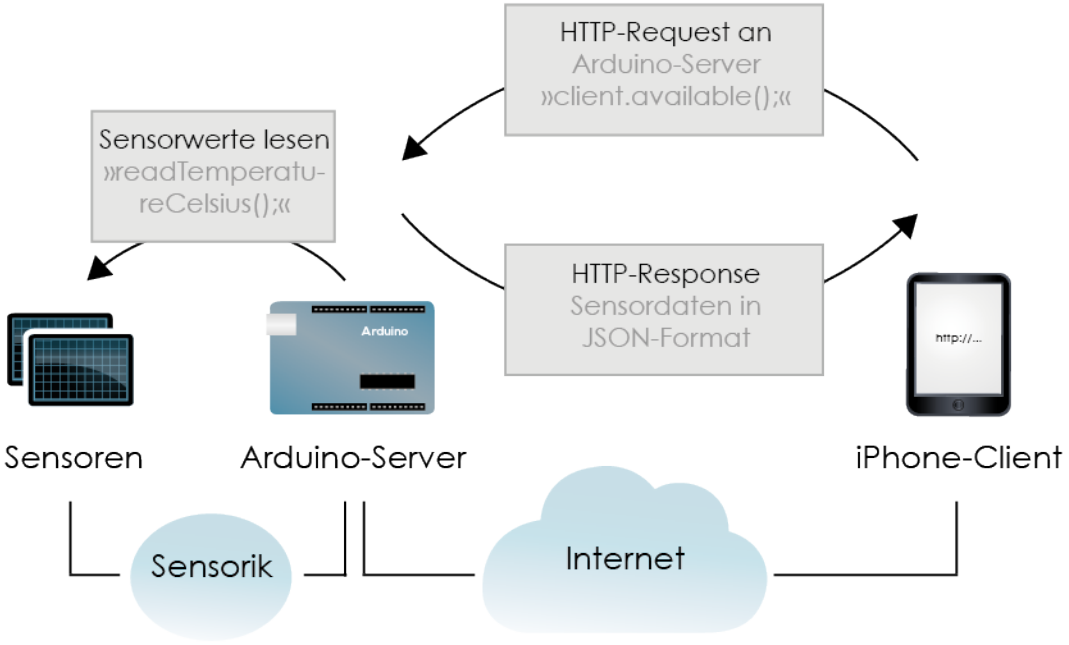


Abbildung 10: Kommunikationsarchitektur - Temperatur-Sensor über iOS-Applikation auslesen

Bei diesem Projekt werden die verarbeiteten Daten über eine iOS-Applikation ausgegeben. Die Besonderheit bei diesem Projekt liegt in der Verwendung von JSON bei Applikationen für iPhones, was seit iOS 5 durch implementierte APIs sehr einfach umsetzen ist. JSON-formatierte Daten ermöglichen einfaches Lesen und Analysieren von Daten durch externe Anwender. Der HTTP-Response muss dahingehend dem JSON-Format angepasst werden und die Temperatur-Variablen, welche die Sensorwerte speichern, mit dem Response übergeben werden.

Die Temperatur-Sensoren werden über die zwei Buchsen D5 und D6 auf dem »Grove Base Adapter« angeschlossen, welche den Pins 5 und 6 auf dem Arduino-Board entsprechen. Für die »1-Wire-Bus«-Technologie stehen Bibliotheken speziell für Dallas-Temperatur-Sensoren zur Verfügung, welche in den Kopfbereich (Header) des Programmcodes inkludiert werden müssen. Zwei Konstanten übernehmen die Pinbelegung der beiden Temperatur-Sensoren. Durch das Deklarieren zweier »OneWire«-Objekte kann eine Kommunikation mit den Sensoren hergestellt werden. Diese werden noch der »DallasTemperature«-Bibliothek zugewiesen und sind somit funktionsfähig. Die Objekte können über eine Read-Funktion ausgelesen und an die Variablen übergeben werden, welche wiederum in dem Response übermittelt werden. In der Read-Funktion werden lediglich die Sensoren ausgelesen und als Fließkommazahl übergeben.

Die Applikation in folgender Abbildung wird nativ über »Xcode[14]« erstellt und verwendet hierzu das »Utility Application Template[15]«, welches über eine Einstellungsebene verfügt,

[14] Eine integrierte Entwicklungsumgebung von Apple, mir der man Anwendungen füt iOS schreiben kann.
[15] Utility Applikacation Template beutetet sinngemäß eine Dienstprogramm-Vorlage

die über ein Flip-Effekt aufgerufen werden kann und der Eingabe von Konfigurationsparametern dient. Hier kann die URL[16] zum Arduino-Server eingetragen werden. Die Applikation kann wie folgt aussehen, je nachdem welche und wie viele Werte gelesen und analysiert werden.

Abbildung 11: Temperature-Sensor iOS Applikation [30]

Genauere Information wie auch der Programmcode ist unter *http://www.raywenderlich.com/38841/* erhältlich. [30]

2.1.2 Hausautomation

Hausautomation oder auch Heimautomation behandelt grundlegend Produkte oder Dienstleistungen, welche im Haus oder in der nahen Umgebung Aktionen auslösen oder Nachrichten ohne einen direkten Eingriff des Besitzers übermitteln. Bei vielen Personen befinden sich Geräte mit Mikrocontroller im Haushalt, welche unter die Rubrik »Heimautomationsgerät« fallen. Das können zum Beispiel Wecker, Rauchmelder, Waschmaschinen und vieles mehr sein. Diese Geräte verwenden für gewöhnlich kein Netzwerk-Kommunikationsprotokoll, worüber diese sich mitteilen oder untereinander kommunizieren können.

In den 70er Jahren kam die Filterkaffeemaschine namens »Mr. Coffee« auf den Markt, die über eine Leiterplatine mit Mikrocontroller verfügte, worüber ein Zeitschaltuhr eingestellt werden konnte. Somit war die Maschine im Stande den Brühprozess ab der eingestellten Zeit zu starten. Insofern hat die Heimautomation schon sehr früh in die Haushalte Einzug erhalten. Heute ist es Bastlern möglich, eine einfache Kaffeemaschine dahingehend zu überarbeiten, dass Netzwerkadapter, Temperatur-Sensoren und Mikrocontroller miteinander so verbunden werden, dass der Brühvorgang mit genau der richtigen Dauer und Temperatur für optimale Ergebnisse durchgeführt werden kann und anschließend dem Nutzer eine SMS zugesandt wird mit der Mitteilung, dass der Brühprozess beendet sei. [31]

[16] Akronym für »Uniform Resource Locator«

Derartige Vernetzungen werden früher oder später von den Herstellern der Geräte selbst und wesentlich günstiger vorgenommen, dennoch sind Vorreiter die Hobbybastler, welche nach Lösungen suchen, um gewisse alltägliche Probleme zu lösen. In naher Zukunft soll ein Standard-Kommunikationsprotokoll entwickelt werden, das die Kommunikation zwischen den Geräten und einem Netzwerk regeln und vereinheitlichen soll. Doch wie lange dieses Protokoll noch auf sich warten lässt, ist ungewiss. Daher liegt es an den Bastlern, derartige integrierte Systeme zu entwickeln. [31]

Des Weiteren gibt es eine Vielzahl an kommerziellen Lösungen, welche die Heimautomations-Kommunikation versucht zu standardisieren. »X10«, eine der ersten großen Firmen in diesem Bereich, entwickelt und fertigt noch immer einfache und relativ günstige Heimautomations-Lösungen. Dabei greifen diese Systeme auf das bestehende Stromnetz im Haus zurück und übertragen über dieses per Pulsecode-Protokoll Daten von der X10-Basisstation an X10-Kommunikationsschnittstellen oder umgekehrt. Doch auch bei diesen Systemen treten immer wieder Probleme mit Signaldämpfung, Prüfsummen und Rückmeldungen auf und haben dahingehend eine größere Verbreitung verhindert. [31]

Ein weiterer großer Schritt Richtung Heimautomation hat Google mit der Idee angekündigt, sein Android-Betriebssystem in Haushaltsgeräte zu integrieren. 2011 wurde auf der Google-IO-Konferenz die offizielle Android Open Accessory API und das Accessory Development Kit (ADK) vorgestellt. Diese sollen die Möglichkeit bieten über das Android-Betriebssystem, Zugriff auf preiswerte Mikrocontroller, Sensoren und Aktuatoren zu erhalten. Der Schwerpunkt liegt hierbei auf dem ADK, welcher als Hardwarespezifikation versuchen soll, die Kommunikation zwischen Geräten zu standardisieren. Ein Android-Betriebssystem soll dann auf die Nachrichten entsprechend reagieren können. Google erhofft sich darüber eine Revolution im Hausautomations-Markt, hierzu müssen allerdings ausreichend Hersteller von elektronischen Geräten sich dieser Spezifikation annehmen. Auch Apple und Microsoft versuchen sich in diesem Markt zu integrieren und haben schon durch kleinere Lösungen und Experimente gezeigt, dass der Trend zu Hausautomationslösungen keinesfalls stagniert. Einige Standards in diesem Bereich existieren bereits und sind im Anhang B »Standards - Hausautomation« nachzulesen. [31]

Statt auf einen Standard zu warten und die Zeit verstreichen zu lassen, bietet das Jetzt, durch günstige Hardware, welche über ein TCP/IP-Protokoll verfügt, die Möglichkeit selbst Hand anzulegen und Geräte für den Eigengebrauch zu entwickeln. Durch die folgenden zwei Anwendungen »Android Türöffner« und »Self-Watering Plant« haben Mike Riley[17] und Randy Sarafan[18] zwei signifikante Projekte im Bereich der Heimautomation entwickelt.

Android Türöffner

Mike Riley zeigt bei diesem Projekt, wie sich mit einfachen mitteln ein Türschloss über ein Android-Smartphone öffnen lässt. Als weiteres Feature wird von der eintretenden Person ein Foto gemacht und an das Smartphone geschickt.

[17] Autor des Buches »Das intelligente Haus« (Riley, 2012)

[18] Entwickler und Autor von Arduino Projekten, zum Beispiel »Self-Watering Plant«

In diesem Projekt wird auf ein günstiges Android-Smartphone (erste Generation), ein Sparkfun IOIO-Board (Jojo) und Relais zurückgegriffen. Riley verwendet das preiswerte Sparkfun IOIO, ein Arduino-kompatibles Board, über welches die Datenverarbeitung und Steuerung per Android einfacher umzusetzen ist, da es ausschließlich für die Steuerung des »PowerSwitch Tails[19]« zuständig ist. Das Smartphone fungiert hierbei als Server, welcher auf eine Öffnungs-Aufforderung, gesendet von einem anderen Android-Smartphone (Tür-Client), reagiert. Wurde die Anforderung des Android-Servers ausgelöst, so erstellt dieses Smartphone ein Foto und schickt dieses per Mail an einen im Voraus definierten Nutzer.

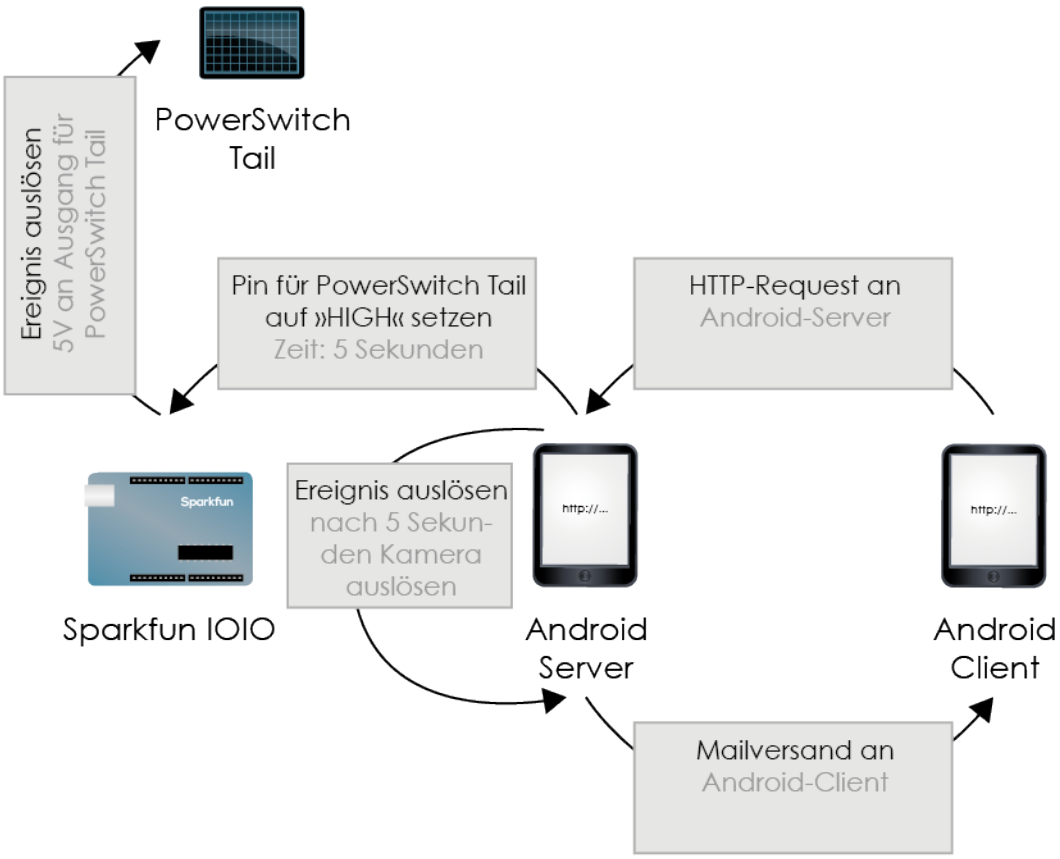


Abbildung 12: Kommunikationsarchitektur – Android-Türöffner

Folgende Komponenten werden bei diesem Projekt verwendet.

- PowerSwitch Tail IIU (Relais Schaltung zum Steuern von 240 V)
- Stromversorgung (5 V DC, 1 A)
- Schaltnetzteil (12 V, 5 A) für den elektrischen Türöffner
- Smartphone mit Android-OS und integrierter Kamera
- Zweipoliges Kabel von Hohlstecker auf JST, um den JST-Anschluss auf das IOIO-Board anzuschließen
- Sparkfun IOIO Board mit JST-90° Anschluss (speziell für Android)
- Elektrischen Türöffner (12 V DC) von Smarthome Electric
- USB-Kabel von A auf Mini-B

[19] ein Relais gesteuertes Netzteil

Als Tür-Client wird ebenfalls von einem Android-Gerät ausgegangen, ob Smartphone, Tablet oder Sonstiges. Der Entwickler arbeitet mit der Eclipse[20]-IDE auf der das Android Development Tools (ADK) Plugin für Eclipse zu implementieren ist. Zusätzlich wird noch die Android-SDK in der Version 1.5 oder höher benötigt.

Bei diesem sehr umfangreichen Projekt wird viel Zeit für den Zusammenbau und die Testphasen der Hardware benötigt. Nach dem Überprüfen der Hardware wird das Android-Smartphone programmiert, damit eine Kommunikation mit dem IOIO-Board von Sparkfun, der integrierten Kamera und dem drahtlosen Netzwerk hergestellt werden kann. Anschließend wird eine einfache Client-Applikation entwickelt, welche als Auslöser dient. Durch das Auslösen wird das IOIO-Board angesprochen, welches wiederum den PowerSwitch Tail aktiviert und dieser anschließend den elektrischen Türöffner auslöst. Daraufhin wird ein Programm auf dem Android-Sever die interne Kamera auslösen und eine Mail mit angehängtem Bild an eine definierte Adresse schicken.

Abbildung 13: System-Aufbau [31]

Die Türöffner-Client-Applikation wird mit einem Button ausgestattet, welcher die URL des Android-Severs aufruft. Ab diesem Aufruf ist der Auslöser für 5 Sekunden lang aktiv. Anschließend wird der PowerSwitch Tail nicht mehr mit Strom versorgt, was ein Schließen des Türmechanismus bewirkt.

Mit diesem Projekt ist es möglich von überall her Personen bei Bekanntgabe in die Wohnung beziehungsweise in das Haus eintreten zu lassen. Aufgrund des sehr komplexen und umfangreichen Programmcodes wird hier auf weitere Informationen und Erläuterungen verzichtet. Bei weiterem Interesse sind in dem Buch »Das intelligente Haus von Mike Riley« alle Programmcodes und Anleitungen nachzulesen. [31]

[20] quelloffenes Programmierwerkzeug

Self-Watering Plant

»Self-Water Plant« ist ein auf Arduino basierendes, autonomes Bewässerungssystem für Pflanzen. Über einen Sensor im Erdreich und eine von Arduino gesteuerte Wasserpumpe lassen sich Pflanzen je nach Feuchtigkeitsgehalt der Erde automatisch bewässern.

Abbildung 14: Bewässerungssystem [32]

Für dieses Projekt werden folgende Hardwarekomponenten benötigt.

- Arduino-Board (zum Beispiel: Arduino-UNO Rev 3)
- Kleine Leiterplatte
- SPDT[21] Relais (5 V DC, 1 A)
- 9 V Batterie Anschluss
- 9 V Batterie
- SPST[22] Toggle-Schalter
- 10 kOhm Widerstand
- Koaxial-Netzanschluss (Größe M)
- elektrische Wasserpumpe (keine Tauchpumpe)
- Wasserbehälter (Wasser-Reservoir)

Der Aufbau dieses Projektes ist wesentlich unkomplizierter als bei den Projekten zuvor. Das Relais dient hierbei als An-/Aus-Schalter für die Wasserpumpe und wird über zwei Kontakte im Erdreich geschalten. Ab einem gewissen Feuchtigkeitsanteil in der Erde besteht eine elektrische Leitfähigkeit zwischen den beiden Kontakten und das Relais schaltet die Wasserpumpe aus. Wird dieser Stromkreis bei Trockenheit unterbrochen, schaltet das Relais im Umkehrschluss die Wasserpumpe an. Der Toggle-Schalter wird für die Spannungsversorgung des Arduinos verwendet, um das gesamte System an- und abschalten zu können. Wie die Hardwarekomponenten verbaut sind, ist in Abbildung 15 zu erkennen. [32]

[21] »Single Pole, Double Throw« ist die englische Bezeichnung für Wechselschalter, wenn ein Stromkreis geöffnet wird, wird der andere geschlossen

[22] »Single Pole, Single Throw« ist die Bezeichnung für einen einpoligen Schalter

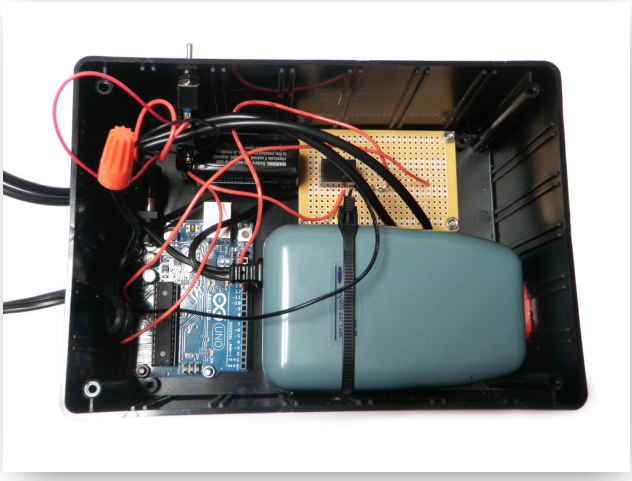


Abbildung 15: Hardware-Bewässerungssystem [32]

Mit diesem System kann ein sehr preiswertes und autonomes Bewässerungssystem realisiert werden. Links zu den Hardwarekomponenten und eine Anleitung stehen unter *http://www.instructables.com/id/Self-Watering-Plant/?ALLSTEPS* zur Verfügung.

Was möglich ist!

Die vorgestellten zwei Projekte bestechen durch ihre Funktionalität und den autonomen Anwendungsfall. Allerdings ist das Anwendungsspektrum sehr vielfältig und es lassen sich so gut wie alle mechanischen und elektrischen Elemente im Haushalt manipulieren oder hacken. Dahingehend gibt es Wasserstandsmesser für eintretendes Wasser, elektronische Wachhunde, welche unbefugtes Eintreten mitteilen, Vogelhäuschen, die einen zu niedrigen Trinkwasser- oder Futterstand per Twitter übermitteln, über das Internet steuerbare Lichtschalter, Rollo-Automation zum automatischen Abdunkeln, oder Sprachausgaben wie auch Steuerungen für den Heimbereich. Dies ist eine kleine Auswahl von bereits realisierten Projekten und tagtäglich kommt eine Vielzahl an neuen Anwendungen und Systemen hinzu.

2.2 Arduino Funktionalität

Die Hardware des Arduino hat sich stetig seit seiner Einführung im Jahr 2005 weiterentwickelt. Da Arduino als Konzept der Kombination aus Hardware und Software entspricht, ist es von großer Bedeutung, ein Verständnis in beiden Bereichen zu erlangen und die Überlappung dieser zu verstehen. In diesem Kapitel wird auf die grundlegende Hardware näher eingegangen. [33]

Auf dem Markt ist eine Vielzahl offizieller Arduino-Boards erhältlich, welche mit der Arduino-Software kompatibel sind. Des Weiteren werden von Mitgliedern der Communities eigene Arduino-kompatible Boards hergestellt und vertrieben. Die beliebtesten unter ihnen enthalten einen USB-Anschluss, welcher die Stromversorgung und die Übertragung der Sketche übernimmt. [5]

Alle Arduino-Boards verfügen über einige spezielle Fähigkeiten und Funktionen. Dennoch sind die Grundkonfigurationen zum größten Teil die selbigen und lassen sich an einem Arduino-UNO am besten aufzeigen.

2.2.1 Grundfunktionen

Folgende Hardware-Komponenten werden überwiegend bei Arduino-Boards verwendet.

- Atmel-Mikrocontroller
- USB-Programmier-/Kommunikations-Interface
- Spannungsregler und Stromanschluss
- Debug, Power, und RX[23]/TX[24] Leuchtdioden
- ICSP[25] Schnittstelle, zum umprogrammieren des Mikrocontrollers [34]

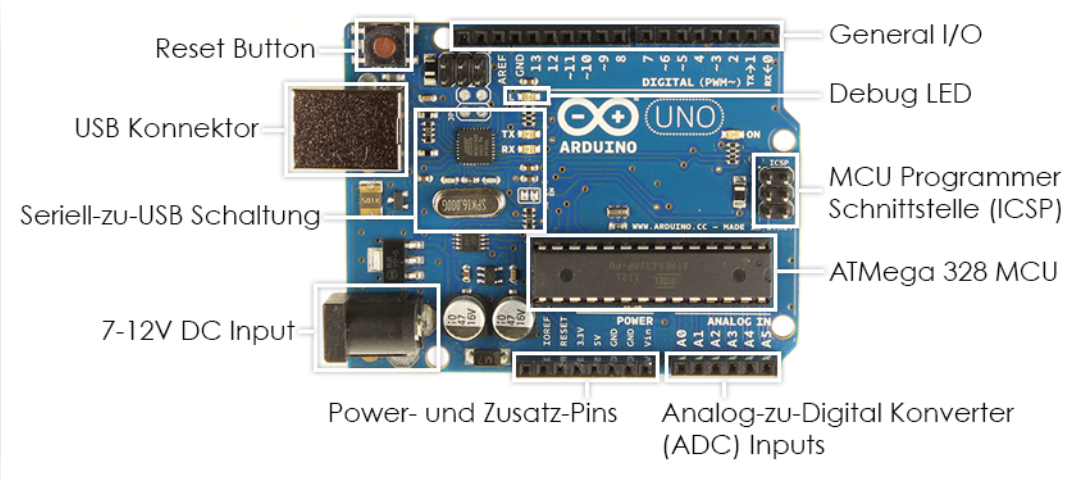


Abbildung 16: Arduino-UNO R3 [35]

2.2.2 Atmel-Mikrocontroller

Das Herz jedes Arduino-Boards bildet die Atmel-Mikrocontroller-Einheit (MCU[26]), welche auf einem Sockel angebracht und bei ein paar wenigen Boards einfach ersetzt oder entnommen werden kann. Dies hat den Vorteil, dass bei defekten Komponenten auf dem Board, der Mikrocontroller auf ein anderes Board übernommen werden kann und wieder funktionsfähig ist. Des Weiteren sind viele Komponenten für den Betrieb eines Boards nach der Entwicklungsphase überflüssig, wodurch wesentlich kleinere und kompaktere Platinen und Schaltkreise realisierbar sind. Die meisten dieser Boards haben einen AVR ATmega-Mikrocontroller implementiert, eingeschlossen der Arduino-UNO R3. Dieses Board verwendet einen ATmega328P-PU, einen 8 Bit Mikrocontroller mit 32 KB[27]

[23] Receiver/Empfänger
[24] Transmitter/Sender
[25] »In Circuit Serial Programming«
[26] Akronym für »Microcontroller-Unit«
[27] KB steht für Kilobyte, 1KB entspricht 1024 Byte

integriertem Flashspeicher, 2 KB RAM Arbeitsspeicher und einem 1 KB großen EEPROM-Speicher. [36], [37]

Das Gehirn bildet die CPU[28], welche die auf dem Flash-Speicher gespeicherten Programmanweisungen abruft und ausführt. Dazu werden Daten aus dem Arbeitsspeicher abgerufen, anschließend bearbeitet und wieder abgelegt. Ähnlich wie der Flash-Speicher ist auch der EEPROM nicht flüchtig und beim Abschalten oder Trennen von der Stromversorgung bleiben darauf abgelegte Daten bestehen. Während der Flash-Speicher zum Speichern von Sketche dient, besteht die Aufgabe des EEPROM darin, Daten zu speichern, welche man durch einen Stromausfall oder beim Zurücksetzen des Gerätes nicht verlieren möchte. Trotz der geringen Speicherkapazität lassen sich einige Anwendungsfälle finden, für die dieser Speicher geeignet ist. Zum Beispiel kann er für sämtliche Netzwerkeinstellungen genutzt werden, die für den Betrieb eines Ethernet-Shields benötigt werden oder zum Speichern von Geräte-Adressen der Sensoren. Ein entscheidender Vorteil bei der Speicherung in diesem Bereich des Mikrocontrollers liegt darin, dass Werte zum Beispiel über das Terminal oder wahlweise über ein Webinterface editiert werden können ohne den Arduino neu zu programmieren. Der Schreibzyklus ist wesentlich langsamer (gegenüber dem Flash-Speicher), daher sollten auf diesem nur relativ persistente Daten gesichert werden. In der Arduino-Entwicklungsumgebung ist eine Bibliothek zum beschreiben und lesen dieses Speichers implementiert. [36], [37]

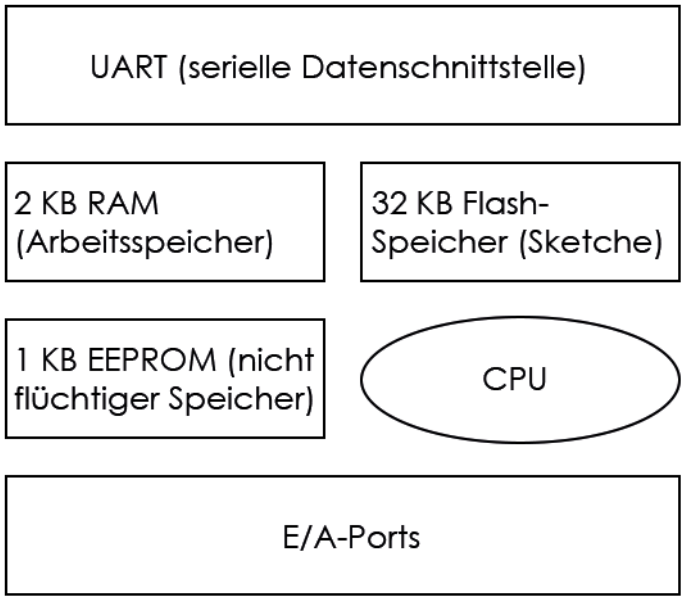


Abbildung 17: Blockdiagramm des ATmega328 [36]

[28] Hauptprozessoreinheit

Folgende Tabelle listet die technischen Daten der MCU.

Produktkategorie	8 Bit Mikrocontroller – MCU
Kern	AVR
Maximale Taktfrequenz	20 MHz
Flash-Speicher	32 KB (0,5 KB für Bootloader)
SRAM	2 KB
On-Chip ADC[29]	Ja
Schnittstellentyp	I2C(TWI), SPI, USART
Betriebsversorgungsspannung	1,8 V bis 5,5 V
Digitale Input-/Output-Pins	14 (6 davon als PWM nutzbar)
Analoge Input-Pins	6

Tabelle 1: ATmega328P-PU [38]

Durch die Arduino-Programmiersprache erhält man Zugriff auf die Mikrocontroller-Peripherien. Diese umschließen die ADCs, Eingangs-/Ausgangs-Pins, Kommunikations-Buses (I2C und SPI) und Serielle-Interfaces. Alle diese Funktionen werden an den Pins bereitgestellt, an denen wiederum Drahtverbindungen oder Shields angebracht werden können. Ein 16-MHz-Keramikresonator ist mit dem ATmega Takt-Pin verbunden, welcher beim Ausführen von Programmbefehlen als Referenz dient. Der Reset-Button dient zum erneuten Ausführen des Programms, ohne dass mit dem Computer neu verbunden werden muss. Eine Debug-LED, welche mit Pin 13 verbunden ist, ermöglicht die Überprüfung der MCU und des ersten Test-Sketches ohne weitere Hardware oder Schaltungen. Die Pinbelegung eines ATmega328 ist im Anhang C einzusehen. [5], [34], [36], [39]

2.2.3 Programmierschnittstellen

Normalerweise werden ATmega-Mikrocontroller-Programme in einem »C/C++«-Dialekt geschrieben. Der vorinstallierte Bootloader erlaubt einen Upload von neuem Code ohne die Verwendung eines externen Hardware-Programmers. Zusätzlich kann der Programmcode auch in Assemblersprache geschrieben und über die ICSP-Schnittstelle auf den Mikrocontroller geladen werden, wodurch der Bootloader umgangen wird. Hierzu bietet Atmel einen AVR-ISP-Programmer an. Die bequeme Programmierung über die USB-Schnittstelle ist dem Bootloader zu verdanken, welcher von Werk aus auf dem ATmega geladen ist. Beim Arduino-UNO und -Mega 2560 ist ein zweiter Mikrocontroller (ATmega16U2 oder 8U2) auf der Leiterplatine angebracht, welcher als Schnittstelle zwischen dem USB-Kabel und den seriellen USART-Pins auf dem Hauptcontroller dient. Frühere Arduino-Boards haben hingegen einen extra FTDI USB-zu-Serial Treiber Chip benötigt. [34], [39]

[29] Analog zu Digital Konverter

2.2.4 General I/O

Der Arduino-UNO verfügt über 14 digitale Pins, die als Eingang oder Ausgang genutzt werden können. Alle Pins können in einem Programm individuell genutzt und adressiert werden. Diese arbeiten dabei mit 5 Volt und verfügen über einen Pull-Up Widerstand von 20 bis 50 kOhm. Des Weiteren verfügen einige Pins über spezielle Funktionen.

- **Pin 0 (RX) und 1 (TX)**
  Diese Pins erlauben das Senden (TX) und Empfangen (RX) von seriellen Daten. Sie sind mit den dazugehörigen Pins des zweiten Mikrocontrollers (ATmega16U2 oder 8U2) verbunden.
- **Pin 2 und 3**
  Mit diesen Pins kann je nach Konfiguration ein Interrupt[30] ausgelöst werden, sollte ein Wert zu niedrig, zu hoch oder ansteigend wie auch fallend sein.
- **Pin 3, 5, 6, 9, 10, 11**
  Diese werden auch PWM genannt und verfügen über einen 8-Bit PWM Output. Dadurch können diese anders als normale digitale Ein- und Ausgänge (0 und 1) Werte in einem Bereich von 0 bis 255 ausgeben (siehe Anhang A: Pulse-Width-Modulation).
- **Pin 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)**
  Diese vier Pins unterstützen, mit der zugehörigen Bibliothek, die SPI-Kommunikation.
- **Pin 13 (L)**
  Dieser Pin ist mit der auf dem Board gekennzeichneten LED (L) verbunden. Dieser kann zum Testen oder Debuggen des Boards verwendet werden.

Der Arduino-UNO verfügt zusätzlich über sechs analoge Eingänge (Pin A0 bis A5) mit einer Auflösung von 10 Bit was wiederum 1024 Abstufungen entspricht. Wie auch bei den digitalen verfügen die analogen Pins über erweiterte Funktionen.

- **Pin A4 (SDA) und A5 (SCL)**
  Diese unterstützen unter der entsprechenden Bibliothek die TWI[31]-Kommunikation.

Des Weiteren verfügt das Board über einen IOREF-Pin für die Referenzspannung, mit welcher der Mikrocontroller arbeitet und einem RESET-Pin, im Falle eines verbauten Reset-Buttons kann über diesen Pin das Board neu gestartet werden. [34], [39]

2.2.5 Stromversorgung

Bei den meisten Projekten im Entwicklungsstatus wird der Arduino mit 5 Volt über das USB-Kabel versorgt. Allerdings kann auch eine externe Stromquelle angeschlossen werden. Für diese Art an Versorgung kann zwischen einem AC-zu-DC Netzteil oder einer Batterie gewählt werden. Die Batterie kann über einen Barrel-Jack-Adapter oder über den Vin-

[30] Unterbrechung des Programmcodes

[31] »Two-Wire-Interface« unterstützt einen 7 Bit breiten Adressraum womit 128 Geräte miteinander verschaltet werden können. Im Gegensatz zu SPI werden hierzu nur 2 Busleitungen benötigt. [40]

(Volt-in) und dem Gnd-Pin (Ground) verbunden werden. Das Board kann mit 6 bis 20 Volt versorgt werden. Bei Spannungen unter 7 Volt kann es allerdings vorkommen, dass der 5V-Pin weniger als die gewünschten 5 Volt liefert und das Board instabil wird. Bei mehr als 12 Volt hingegen kann der Spannungsregler überhitzen und somit das Board beschädigen, daher wird meist ein Spannungsbereich zwischen 7 und 12 Volt angegeben. Das Arduino-Board verfügt über integrierte 5 und 3,3 Volt Regulatoren. 5 Volt wird für die gesamte Logik auf dem Board benötigt. Setzt man zum Beispiel einen digitalen Pin auf »HIGH«, so liefert dieser am Ausgang 5 Volt. Der 3,3 Volt Pin ist für die Versorgung von Shields und externen Verbrauchern zuständig, welche lediglich 3,3 Volt benötigen. [34], [39]

2.3 Sensorik

Damit Arduino die physikalische Welt erfassen kann, benötigt es Sensoren, welche auf Ereignisse reagieren und diese in elektrische Signale umwandeln. Die Verfügbarkeit des zu messenden elektrischen Signals ist daher abhängig von der Art des Sensors und der Anzahl zu messender Daten. Ein Teil der erhältlichen Sensoren besteht aus einem Material, welches seine elektrische Eigenschaft auf Änderungen in der physikalischen Welt hin verändert. Des Weiteren existieren auch wesentlich komplexere Sensoren oder Sensorschaltungen mit eigenem Mikrocontroller, welche die Daten erst verarbeiten ehe sie diese an der Arduino weiterleiten. Folgende Methoden werden von Sensoren verwendet um Daten und Informationen für den Arduino bereitzustellen. [5], [34]

- **Digital An/Aus**
  Bewegungs-, Tilt-Sensoren und viele Weitere geben ausschließlich an, ob ein Sensor was erfasst hat oder auch nicht. Hierzu wird zum Beispiel »1« (5 V) und »0« (0 V) angegeben um einen Zustand zu beschreiben.
- **Analog**
  Diese Art an Sensor liefert ein analoges Signal, worin die Spannung proportional zum abgerufenen Wert ist. Hiermit können zum Beispiel Lichtintensität, Stärke einer Vibration oder Beschleunigung gemessen werden.
- **Impulsbreite**
  Hierbei werden Daten in Form eines Impulses bereitgestellt, deren Länge proportional zum Abstand ist. Dabei wir die Dauer des Impulses gemessen und übergeben.
- **Seriell**
  Derartige Sensoren kommunizieren über ein serielles Protokoll zum Übermitteln der Daten. Die meisten Boards verfügen standardgemäß über nur einen seriellen Port (siehe Kapitel 2.4.1 »UART und USART«).
- **Synchrone Protokolle**
  Durch dieses Protokoll kann ein Arduino mit externen Sensoren oder Module kommunizieren (siehe Kapitel 2.4.3 »I2C und SPI«). [5], [34]

2.3.1 Bewegungs- und Beschleunigungs-Sensor

Diese Sensorart ist für die Aufnahme von Beschleunigung und Neigung zuständig. Durch piezokeramische Plättchen nimmt dieser Kapazitätsänderungen oder Druckänderungen wahr und wandelt diese in ein elektrisches Ausgangssignal um. Somit können Werte für

verschiedene Richtungsänderungen übergeben werden. Durch die Spielekonsole »Wii« hat der Bedarf an Beschleunigungssensoren stark zugenommen worauf ein Arduino-Klon »Wii-Nunchuk-Controller« entwickelt wurde, der auf die Bewegungsänderungsmessung ausgerichtet ist. Ein passiver Infrarot-Sensor (PIR) hingegen ändert den Wert an einem digitalen Pin des Arduino-Boards und kann somit nur »1« oder »0« als Wert liefern. [5], [8]

2.3.2 Licht-Sensor

Hierbei wird ein lichtempfindlicher Widerstand eingesetzt, welcher seinen Widerstandswert in Abhängigkeit zur Lichtintensität ändert. Um möglichst exakte Werte und viele Abstufungen zu bekommen, wird der Sensor an einen analogen Pin am Board angeschlossen wodurch ein Wertebereich von 0 bis 1023 zur Verfügung steht. Meist müssen die resultierenden Werte anschließend für die Weiterverarbeitung noch in brauchbare Werte umgewandelt werden, hierzu stellt Arduino gewisse Funktionen zur Verfügung. Um genaue Resultate zu erlangen, muss die Wahl des lichtempfindlichen Widerstandes den Begebenheiten des Anwendungsbereiches entsprechen. [5], [41]

2.3.3 Ultraschall-Sensor

Dieser Sensor wird zur Abstandsmessung verwendet. Hierbei können zum Beispiel Abstände zu einem Gegenstand oder sich einer des Sensors nähernden Person gemessen werden. Ein hierfür häufig verwendeter Sensor ist der »Parallax-PING))) Ultrasonic«-Sensor, welcher Abstände von 2 cm bis zu 3 m messen kann. Dabei misst der Sensor die Zeit, die der Schall benötigt, um vom Sensor zum Objekt hin und wieder zurückzukommen. Die Methode, die bei dieser Sensorart verwendet wird, ist die Impulsbreite, wie in Kapitel 2.3 beschrieben. Die Länge des Impulses ist dabei proportional zur Strecke. Ein weiterer Sensor dieser Rubrik ist der »MaxBotix EZ1«, der wesentlich einfacher in ein System zu integrieren ist. [5], [42]

2.3.4 Vibrations-Sensor

Ein Piezo-Sensor reagiert auf eine physikalische Belastung dahingehend, dass dieser eine Spannung erzeugt. Die Höhe der Spannung ist abhängig von der Stärke der Belastung. Um diese Werte messen zu können, muss der Sensor ebenfalls an einem analogen Eingang des Boards angebracht werden. Ein Piezo-Sensor kann auch als Ausgang dienen und die Membran durch Spannung zum Schwingen verleiten, um so einen Ton zu erzeugen. [5], [43]

2.3.5 Temperatur-Sensor

Der Temperatur-Sensor »LM35« ist ein häufig verwendeter transistorähnlicher Sensor. Dieser erzeugt eine Analogspannung, welche zur Temperatur proportional ist. Er hat dabei eine Abweichung von zirka 0,5 Grad Celsius. [31]

2.3.6 Rotationswinkel-Sensor

Diese werden auch Gyroskop genannt und messen den Rotationswinkel. Hierbei ist die Analogspannung proportional zum Winkel des Gyroskops. Einige derartige Sensoren bieten ihre Daten und Werte über I2C an, dazu mehr im nächsten Kapitel. Es können bis zu drei Achsen damit gemessen werden, x-, y- und z-Achse. Dabei wird die Geschwindigkeit der Änderung einer bestimmten Achse zum aktuellen Zeitpunkt gemessen. Um daraus einen Winkel nach mehreren Änderungen noch bestimmen zu können, müssen alle Veränderungsraten in einem bestimmten Zeitraum aufsummiert werden. [5], [44]

2.3.7 E-Health[32]-Sensoren

Auch Sensoren für den medizinischen Bereich oder zur Überwachung von Vitalfunktionen können in Verbindung mit einem Arduino eingesetzt werden. Hierbei werden Sensorhacks oder eigens dafür entwickelte Hardwarekomponenten bereitgestellt, mit denen sich zum Beispiel der Sauerstoffgehalt im Blut, Puls, Körpertemperatur, Blutdruck und vieles mehr erfassen lassen.

2.4 Bus-Systeme

Bus-Systeme bezeichnen Leitungssysteme, die zum Austausch von Daten oder Energie zwischen verschiedenen Hardwarekomponenten dienen. Folgend werden verschiedene Systeme vorgestellt, die durch das Arduino-Board unterstützt werden.

2.4.1 UART und USART

Bei UART handelt es sich um ein eigenständiges elektronisches Bauteil, wie zum Beispiel einen Chip oder um einen Funktionsblock, welcher in Mikrocontrollern verwendet wird. Die zugehörige Schnittstelle ermöglicht das Senden und Empfangen von Daten und bildet standardgemäß die serielle Schnittstelle an Computern und Mikrocontrollern. Eine bidirektionale UART-Kommunikation benötigt immer zwei Datenleitungen, eine zum Senden und die andere zum Empfangen von Daten. Zusätzlich wird noch eine Masseleitung benötigt. Arduino-Boards verfügen, je nach Bauart, über mindestens einen seriellen Port. Hierbei findet die Kommunikation über die Pins 0 (RX) und 1 (TX), wie auch mit dem Computer über USB statt. Wird eine serielle Verbindung verwendet, wie zum Beispiel den in der Entwicklungsumgebung integrierten »Seriellen-Monitor«, können die Pins 0 und 1 nicht weiterhin für digitalen Eingang beziehungsweise Ausgang genutzt werden. Das Arduino-Mega hingegen verfügt über drei weitere serielle Ports. Der »Serielle-Monitor« kann für die Kommunikation zwischen Computer und Arduino-Board genutzt und Daten darüber ausgegeben werden. Bei dem »Seriellen-Monitor « muss dieselbe

[32] Anwendungen von elektronischen Geräten zur medizinischen Versorgung oder Überwachung im Gesundheitswesen

Baudrate[33] eingestellt werden, wie im Sketch gewählt wurde, da auf Empfänger- und Senderseite bei einer Datenübertragung stets dieselbe Baudrate benötigen wird. [39], [46]–[48]

Viele AVR-Mikrocontroller verfügen über eine Erweiterung namens »USART«, welche für »Universal Synchronous and Asynchronous serial Receiver and Transmitter« steht. Hierbei wird UART um die Fähigkeit der synchronen Datenübertragung erweitert. Somit kann dieser auch als SPI-Master eingesetzt werden. Auch der Arduino-UNO R3 verfügt über einen USART. Erklärung zu SPI folgt im darauffolgenden Kapitel. [46]

Es wurden weitere Standards entwickelt, um Informationen, anders als bei USART, zwischen einem Mikrocontroller und Sensoren auszutauschen. Durch entsprechende Bibliotheken vom Hersteller ist die Implementierung für ein Arduino einfach gehalten. Die Wahl zwischen den Systemen wird meist schon durch die Bauelemente oder Chips festgelegt, welche man zu verwenden gedenkt. Die verschiedenen Bus-Systeme verwenden unterschiedlich viele Leitungen zur Adressierung eines Sensors und zum Datenaustausch. Allerdings wird bei der Angabe der Systeme in Bezug auf die Leitungs- beziehungsweise Anschlussvarianten die Masse nicht mitgezählt. So enthält zum Beispiel ein »1-Wire Bus-System« nicht nur die eine Leitung, sondern ebenfalls eine Weitere für Masse.

2.4.2 1-Wire

»1-Wire« ist ein einfaches und dennoch leistungsstarkes Bus-System, bei dem über eine Signalleitung serieller Datentransfer und Adressierung ausgeführt werden kann und das auch bei bis zu 100 angeschlossenen Geräten, Sensoren oder Teilnehmern. Besonders eignet sich dieses System für den Einsatz im Sensorikbereich wie der Temperaturmessung, Akkuüberwachung und vielem mehr. Die Datenübertragung erfolgt nach dem One-Master/Multi-Slave-Prinzip, bei dem jeder Slave mit einer 64-Bit-ROM-ID adressiert werden muss. Des Öfteren sind die Slaves allerdings schon von Werk aus mit einem eindeutigen Bezeichner versehen. Im Bereich der Gebäudeautomatisierung sind »1-Wire«-Sensoren aufgrund des sehr günstigen Preises, der einfachen Verkabelung und der sehr genauen Messung von Umweltdaten sehr interessant. Zudem reicht meist eine parasitäre Stromversorgung aus, wodurch eine dritte Leitung für die Stromversorgung entfällt, da der Sensor seine Spannung aus der Datenleitung beziehen kann. [29], [28]

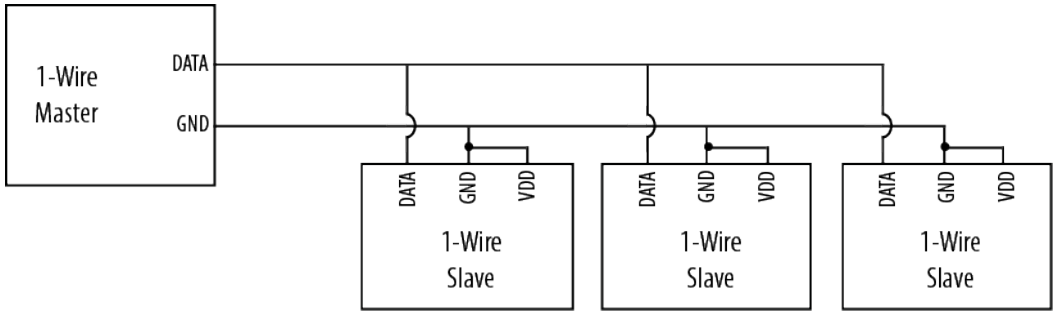


Abbildung 18: Ein 1-Wire Master mit ein oder mehreren 1-Wire Slaves

[33] Die Baudrate gibt die Anzahl der Symbole pro Zeiteinheit an. 1 Baud bedeutet, dass 1 Symbol pro Sekunde übertragen wird. [45]

Anders als bei I2C und SPI sind keine Bibliotheken in der Arduino-Entwicklungsumgebung standardgemäß vorhanden und müssen nachträglich implementiert werden. Ein für Arduino kompatibler Hersteller im Sensorikbereich ist Maxim (früher Dallas) mit gleichnamiger Bibliothek.

2.4.3 I2C und SPI

Die Begriffe »I2C« und »SPI« sind im Laufe der letzten Kapitel des Öfteren aufgetreten. Sensoren unterstützen oftmals beide Protokolle und überlassen somit dem Anwender die Art der Implementierung. Um Verständnis über die beiden Standards zu erlangen, folgt nun eine einführende Beschreibung dieser beiden Protokolle.

I2C

I2C ist ein von Phillips entwickelter serieller Datenbus, welcher aus lizenzrechtlichen Gründen von Atmel auch als »TWI« (Two-Wire-Interface) bezeichnet wird.
Für Bauelemente dieser Art werden nur zwei Leitungen, exklusiv Masse, benötigt. Somit handelt es sich hierbei um ein »2-Wire Bus-System«. Wie bei »1-Wire Bus-Systemen« ist es ebenfalls möglich, mehrere Slaves über diese zwei Leitungen zu betreiben. Allerdings ist die Datenrate eine geringere als bei SPI und es ist immer nur eine Kommunikation in eine Richtung möglich. Sollte eine bidirektionale Verbindung stehen, so verlangsamt sich die Datenrate noch weiter. Des Weiteren wird für die Inbetriebnahme solcher Komponenten jeweils ein Pull-Up-Widerstand benötigt, um eine zuverlässige Datenübertragung zu gewährleisten. I2C ist daher für Sensoren geeignet, welche nur eine geringe Datenrate benötigen oder zum Koppeln von mehreren Arduino-Boards, um zum Beispiel weitere Eingang-/Ausgang-Pins zur Verfügung zu haben. Diese Art von Verbindung nennt man auch »Multiboard-Anwendung«.

Ein I2C-Bus verfügt über zwei Anschlüsse namens »SCL« und »SDA«. Diese sind bei Arduino-Standardboards über die Pins A5 und A4 erreichbar. Pin A5 ist hierbei für SCL zuständig und liefert das Taktsignal (Clock) und Pin A4 hingegen für SDA und übernimmt den Datentransfer. Arduino bildet das Master-Modul und ist für die Koordinierung des Informationsaustausches zwischen den angeschlossenen Geräten (Slaves) zuständig. Es kann nur einen Master geben, welcher die anderen Chips steuert. Aufgrund der gleichen Pinbelegung der Slaves auf dem Master müssen diese über eine Adresse identifizierbar sein. Diese kann entweder konfiguriert werden, wenn zum Beispiel vom gleichen Typ Sensor mehrere angeschlossen sind, oder sie werden direkt mit einer festen Adresse ausgeliefert für gewöhnlich ein 7-Bit-Wert. [5], [34]

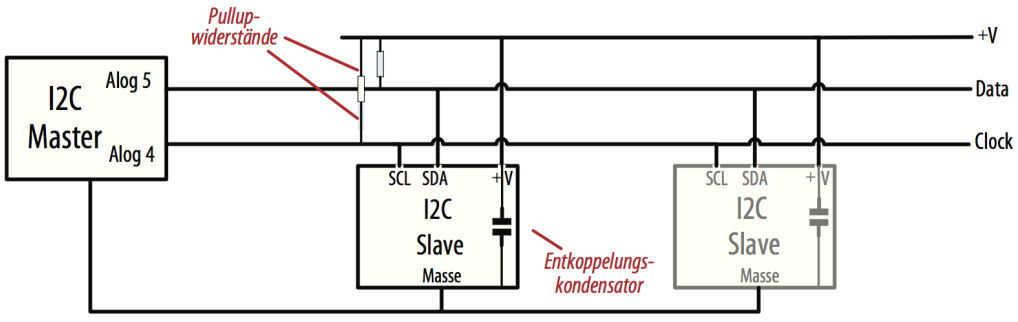


Abbildung 19: Ein I2C-Master mit ein oder mehr I2C-Slaves [5]

SPI

Anders als I2C verwendet SPI separate Leitungen zum Senden und Empfangen von Daten. Des Weiteren verfügt diese Variante auch über eine zusätzliche Leitung, um die jeweiligen Slaves anzusprechen. Daher werden keine zuzüglichen Adressen benötigt, allerdings eine weitere Verbindung. Generell ist SPI einfacher zu konfigurieren und verfügt über eine wesentlich höhere Datenrate bei bidirektionaler Kommunikation als I2C. Es werden drei Pins für eine Master-Slave-Verbindung benötigt. [5]

Zusätzlich verfügt der Slave über den »SS[34]«-Anschluss zum direkten Ansteuern des Slaves. Der Master hingegen benötigt für jeden einen eigenen Pin, welche fortlaufend mit SS1, SS2, SSn bezeichnet werden. Hierbei handelt es sich um ein »n-Wire Bus-System«. Daher gibt die Formel 3 + n die Anzahl der Anschlüsse an, welche am Master benötigt werden, dabei steht »n« für die Anzahl der Slaves. Folgende Anschlüsse sind daher relevant. [5], [34]

- **Serial Clock (SCLK)**
  Ein Signal zum Synchronisieren bei der seriellen Datenübertragung mit dem Empfänger, somit weis dieser, wann der Eingang zu lesen ist.
- **Master Out Slave In (MOSI)**
  Wird zum Senden der seriellen Daten vom Master zum Slave benötigt.
- **Master In Slave Out (MISO)**
  Wird zum Senden der seriellen Daten vom Slave zum Master benötigt.
- **Slave-Select (SS)**
  Zum Ansteuern der einzelnen Slaves. Wird der Wert des jeweiligen Slaves auf »LOW« gesetzt, ist eine Verbindung zu diesem aufgebaut.

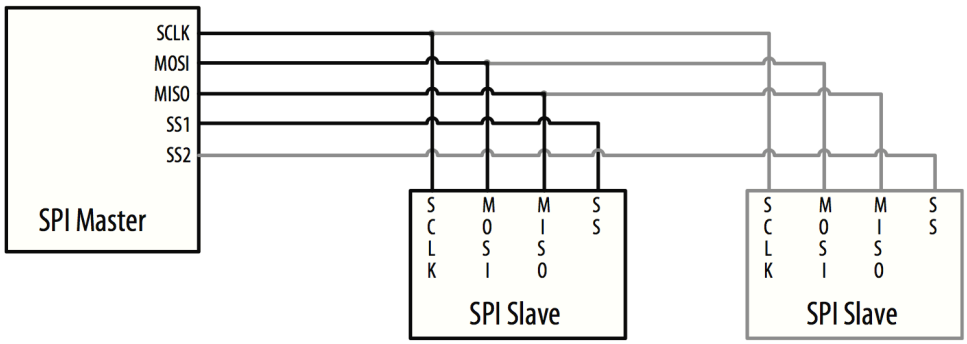


Abbildung 20: Signalanschlüsse für SPI-Master und –Slaves [5]

Vergleich I2C und SPI

Viele Geräte wie Beschleunigungssensoren, digitale Potenziometer und Displays sind in beiden Varianten erhältlich. Die Auswahl der Variante muss dahingehend spezifisch des Anwendungsgebietes und der Benutzerfreundlichkeit getroffen werden. In der folgenden Liste sind die Vorteile der genannten Protokolle noch einmal aufgelistet.

[34] Slave-Select

SPI Vorteile	I2C Vorteile
Höhere Datenrate	Benötigt nur zwei Leitungen
Einfachere Implementierung für Arduino	Arduino Hardware-Unterstützung
Arduino Hardware-Unterstützung	

Tabelle 2: SPI and I2C Comparison [34]

2.5 Datenverarbeitung

Unter dem Begriff »Datenverarbeitung« wird das Anwenden von Algorithmen verstanden. Diese verwenden Daten, um andere Daten abzurufen, Operationen durchzuführen und anschließend wieder auszugeben. Der Ablauf dieser Verarbeitung entspricht dem EVA[35]-Prinzip.

2.5.1 EVA- und EVA(S)-Prinzip

EVA-Prinzip

Hierbei handelt es sich um die Datenverarbeitung ausschließlich im Arduino. Sensoren lesen die Werte aus der physikalischen Welt, der Controller verarbeitet diese und ein Aktuator stellt zum Beispiel die verarbeiteten Werte dar oder führt eine mechanische Bewegung aus. [2]

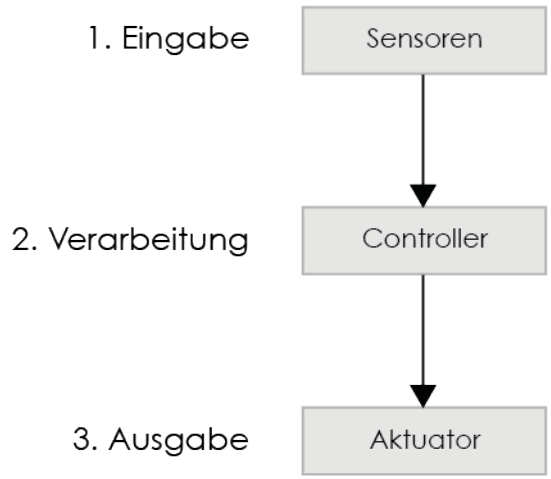


Abbildung 21: EVA-Prinzip

[35] Akronym für Eingabe, Verarbeitung, Ausgabe

EVA(S)-Prinzip

Anders als beim EVA-Prinzip ist hierbei noch eine Speicherung der Daten vorgesehen. Eine Verarbeitung und Initialisierung der Sensorik findet ebenfalls im Mikrocontroller statt, allerdings wird dieser Vorgang nicht separat gelistet in das Schema mit aufgenommen. Die Speicherung kann einerseits auf Seiten eines Arduinos durch die Ergänzung einer SD-Karte geschehen oder andererseits über eine Datenbank, wonach das Prinzip wie folgt dargestellt werden kann.

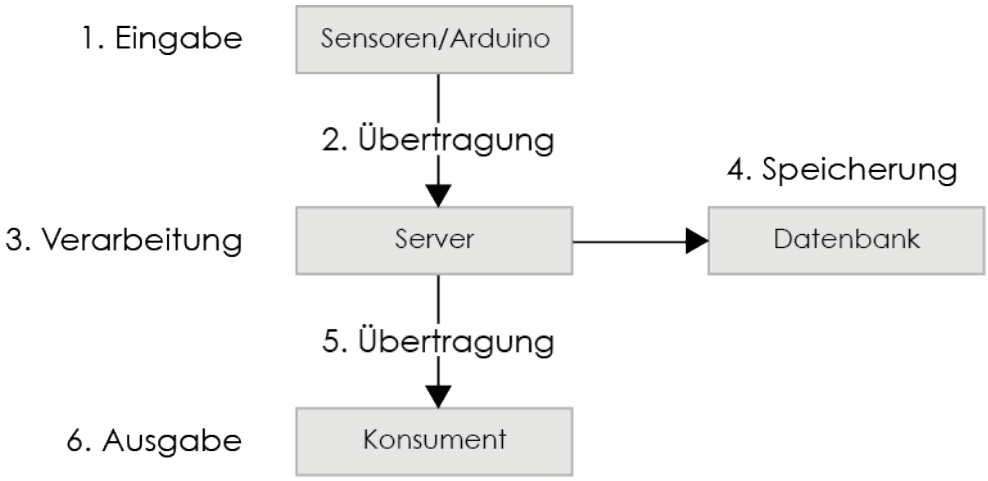


Abbildung 22: EVA(S)-Prinzip [49]

1. **Eingabe**
   Daten werden entweder von Sensoren erfasst, durch Benutzereingaben erzeugt oder durch physikalische Manipulation von Aktuatoren gemessen.
2. **Übertragung**
   Daten werden nach einem definierten Verfahren an einen Server via WLAN oder LAN geschickt.
3. **Verarbeitung**
   Der Server verarbeitet die Daten und bereitet diese in das vom Entwickler vorgesehene Format auf.
4. **Speicherung**
   Dies ist ein optionaler Schritt, bei dem die Daten zum Beispiel auf einer Datenbank für eine zukünftige Nutzung gespeichert werden. Je nach Anwendungsfall können die Daten auch auf einer Speicherkarte auf dem Arduino gesichert werden. Hierbei würde die Kommunikation nicht vom Server, sondern vom Board ausgehen.
5. **Übertragung**
   Die für den Konsumenten bereitgestellten Daten werden an diesen übertragen.
6. **Ausgabe**
   Je nach Anwendung werden die Daten grafisch, akustisch oder eventuell auch mechanisch an den Konsumenten ausgegeben. [49]

2.5.2 Struktur eines Arduino-Sketches

Nach dem EVA-Prinzip findet die Verarbeitung des Programmcodes im Arduino-Controller beziehungsweise im Mikrocontroller statt. Damit ein Sketch lauffähig ist, benötigt dieser zwei programmtechnische Konstrukte, bei denen es sich um sogenannte

Funktionen handelt, die den Rahmen des Programmcodes bilden. Die Initialisierung des Boards für die Anwendung wird in der `setup`-Funktion vorgenommen und nur einmalig nach dem Programmstart ausgeführt. Der Code, welcher kontinuierlich ausgeführt werden soll, wird in die `loop`-Funktion geschrieben. Nach dem Hochladen des Programmcodes auf das Board, werden die Instruktionen nacheinander abgearbeitet, wobei automatisch nach der `setup`-Funktion in die `loop`-Funktion gesprungen wird. Aus der C-Programmierung ist der Ausgangspunkt `main()` bekannt, welcher bei Arduino scheinbar nicht benötigt wird. Allerdings ist dieser durch die Arduino-Build-Umgebung versteckt und daher dennoch vorhanden. Durch eine vom Build-Prozess erzeugte Zwischendatei (siehe Quellcode 1) wird dieser allerdings im Code nicht benötigt.

```
int main(void) {
    init();

    setup();

    for(;;)
          loop();
    return 0;
}
```

Quellcode 1: Durch Build-Prozess erzeugte Zwischendatei [5]

Durch die `init`-Funktion wird die Arduino-Hardware initialisiert. Anschließend wird die `setup`-Funktion des Sketches aufgerufen und danach die `loop`-Funktion fortlaufend ausgeführt, da diese in einer niemals endenden `for`-Schleife eingeschachtelt ist. [2], [5]

Der Sketch kann in drei, für die Programmierung relevanten Bereiche unterteilt werden.

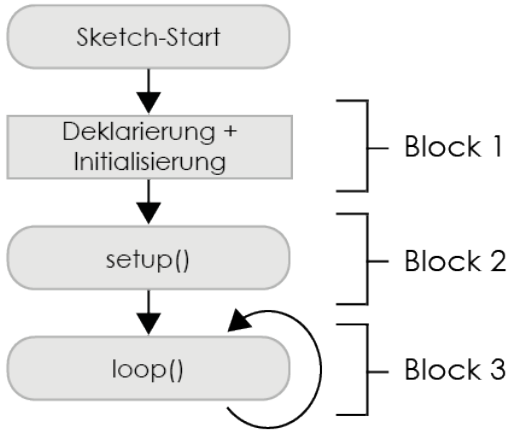


Abbildung 23: Grundlegende Sketch-Struktur [2]

Block 1: Deklarierung und Initialisierung
In diesem Block, oberhalb der `setup`-Funktion, werden externe Bibliotheken über eine `#include`-Anweisung eingebunden. Zudem können hier globale Variablen deklariert werden, die innerhalb des gesamten Sketches verfügbar sind. Bei der Deklarierung wird der Datentyp der Variablen festgelegt, während die Initialisierung der Variablen einen Wert zuordnet.

Block 2: Setup-Funktion
Hier werden die einzelnen Pins des Mikrocontrollers programmiert. Dabei wird festgelegt, welche der Pins als Ein- beziehungsweise Ausgang arbeiten sollen. Bei Sensoren zum Beispiel müssen Signale an den Pin übertragen werden. Dahingehend ist dieser als Eingang festzulegen.

Block 3: Loop-Funktion
Die `loop`–Funktion ist eine Endlosschleife, versehen mit der Logik, welche kontinuierlich Sensoren abfragt oder Aktuatoren steuert. [2]

Programmstruktur

Die Programmstruktur umfasst Kontrollstrukturen, welche die Implementierung von Algorithmen zum Steuern des Programmflusses beinhalten. Des Weiteren sind die arithmetischen Operationen wie auch Zuweisungen von Werten zu Variablen und Funktionsaufrufe darunter zu verstehen. Programmwerte können in zwei Kategorien, Variablen und Konstanten unterteilt werden. Konstanten sind symbolische Namen für konstante Werte, welche eine feste Bedeutung haben. Variablen hingegen sind Ausdrücke, welche einen Wert speichern können.

Um eine logische Trennung und Wiederverwendbarkeit von Abschnitten der Programmstruktur zu erlangen, werden benötigte Blöcke in sogenannte Funktionen unterteilt. Dadurch wird vermieden, dass der gleiche Programmcode mehrmals geschrieben werden muss, wenn dieser derselben Funktionalität untersteht. Hierzu werden die Blöcke als Funktionen abgekapselt und können aus dem Code heraus aufgerufen und durch Parameter, Werte übergeben werden. Bibliotheken bieten bestimmte Funktionalitäten an und werden über definierte Funktionsaufrufe angesprochen. Diese Funktionalitäten können Schnittstellen zu Sensoren und Aktuatoren wie auch arithmetische Berechnungen sein. Diese Zusammenfassung an Funktionen wird als Bibliothek-Schnittstelle oder API[36] bezeichnet. [9]

2.5.3 Variablen, Datentypen und Konstanten

Programme werden unabhängig von Arduino meist in C/C++ oder verwandte Sprachen wie Java geschrieben. Allesamt weisen eine ähnliche Syntaxform auf, allerdings bezieht sich Arduino als Programmiersprache auf C/C++, da der Compiler bereits über eine Teilmenge der Funktionalitäten dieser Sprache verfügt.

Eine große Anzahl an Arduino-Programmen arbeitet mit Daten, die im internen Speicher abgelegt oder durch die Eingang-Pins eingelesen werden. Diese müssen in Variablen gespeichert werden, damit der Programmierer systematisch darauf zugreifen kann. Eine Variable besteht dabei immer aus einem Datentyp, einem Variablennamen und einem Wert. Eine Variablendeklaration sieht hierbei wie folgt aus.

[36] Akronym für »Application Programming Interface« und bedeutet eine Schnittstelle zur Anwendungsprogrammierung

```
Datentyp Variablenname = Wert;
```

Quellcode 2: Variablendeklaration

Datentyp

Neben dem Variablennamen verfügt die Variable über einen Datentyp. Arduino kann mit verschiedenen Variablentypen arbeiten, um Werte bestmöglich repräsentieren zu können. Definiert wird der Typ der Variablen dahingehend über einen Arduino-Datentyp. Dieser legt den erlaubten Wertebereich und die möglichen Operationen, welche mit der Variablen vorgenommen werden können, fest. Bei der Deklarierung der Variablen muss feststehen, welcher Wertebereich erforderlich ist, da die verschiedenen Bereiche dahingehend unterschiedlich viel Speicherplatz in Anspruch nehmen. Jeder Datentyp ist daher genau für Art und Bereich von Werten definiert und muss spezifisch dem Anwendungsfall zugeordnet werden. Ein digitaler Zustand zum Beispiel verfügt ausschließlich über zwei Werte, ein Text hingegen über mehrere Zeichen. Im Anhang D sind die wichtigsten Datentypen und deren zugehöriger Wertebereich aufgelistet.

Konstanten

Eine besondere Art von Variablen sind die Konstanten, deren Wert im Programmablauf nicht verändert werden kann. Aufgrund der statischen Art belegen diese meist auch keinen Speicherplatz, außer wenn es sich bei den Konstanten um eine Tabelle (Array) oder Zeichenkette handelt. Daher werden dieser Variable konstante Werte zugeordnet, zum Beispiel die Nummer eines Pins, Konfigurationsparameter oder bestimme Kommandos für externe Sensoren. Des Weiteren ist für jeden anderen Programmierer durch die Datentyp-Deklarierung ersichtlich, um welchen Variablentyp es sich handelt. [2], [5], [8], [9]

2.5.4 Testen und Debuggen

Das Testen und Debuggen von Soft- und Hardware wird in verschiedenen Stadien der Programmierung vorgenommen. Während der Codeprogrammierung kann der Entwickler in der Entwicklungsumgebung die Syntax mithilfe des Compilers prüfen. Diese Art von Prüfung lässt allerdings keine Rückschlüsse auf den korrekten Ablauf des Programms oder fehlerhafte Strukturen zu. Dies ist nur direkt auf dem Board mit den zugehörigen, externen Komponenten möglich. Bei umfangreichen Projekten mit mehreren Sensoren und Aktuatoren ist eine Prüfung durch kleine Prüfprogramme sinnvoll, vorausgesetzt die Komponenten wurden fachgerecht angeschlossen. Bei der Inbetriebnahme des Programmcodes mit allen Hardwarekomponenten treten oftmals noch Fehler auf beziehungsweise führt das Board nicht die gewünschten Aktionen aus. An dieser Stelle werden dann Funktionen und Aktionen des Programms gezielt überprüft.

Über die serielle Schnittstelle lassen sich einzelne Variablenwerte und Signalzustände innerhalb des Programmablaufs anzeigen. Damit die Prüfwerte vom Entwickler beziehungsweise Programmierer überprüft werden können, werden diese im Seriellen-Monitor ausgegeben.

Serieller-Monitor

Der Serielle-Monitor ist in der Entwicklungsumgebung von Arduino implementiert und ist ein nützliches Werkzeug für den Programmierer zur Fehlersuche. Dieser erlaubt die Anzeige, der über den seriellen Port übertragenen Daten. Darüber hinaus lassen sich nicht nur Daten empfangen, sondern auch vom Computer an das Arduino senden. Auf Sender und Empfängerseite muss hierbei dieselbe Baudrate eingestellt werden. Einerseits geschieht das im Programmcode über folgende Anweisung.

```
void setup() {
  Serial.begin(9600);
}
```

Quellcode 3: Konfiguration der seriellen Schnittstelle, Baudrate 9600

Der Serielle-Monitor muss in diesem Beispiel ebenfalls mit einer Baudrate von 9600 eingestellt werden, um eine fehlerfreie Kommunikation zu gewährleisten. [8]

Debuggen

Wie zuvor erwähnt, kann trotz einer korrekten Kompilierung ein Fehler im Programmcode vorhanden sein. Dieser ist dann nicht auf die Syntax zurückzuführen als vielmehr auf einen falschen Programmablauf. Überwiegend bei komplexen Projekten findet sich ein Fehler in der Ausführung des Codes, welcher schwer zu lokalisieren ist. Daher müssen die Komponenten auf korrekte Funktionsweise durch kleine, eigene Programmcodes getestet werden. Hierzu müssen die Daten im Programm sichtbar gemacht werden und dies geschieht, wie erwähnt, über die serielle Schnittstelle. Die Ausgabe der Daten erfolgt anschließend über den Seriellen-Monitor oder ein Hyperterminal-Programm. In der Anwendung bedeutet es, den Code um weitere Debug-Ausgaben zu erweitern, um so gezielt den Ursprung des Fehlers lokalisieren zu können. Es empfiehlt sich zudem eine Variable zu deklarieren, mit der über eine `if`-Anweisung alle seriellen Ausgaben »Ein« oder »Aus« geschaltet werden können. [5], [8], [9]

2.6 Bibliotheken

In den vorhergegangen Kapiteln war des Öfteren die Sprache von Bibliotheken und deren Einsatz. Um den Hintergrund zu verstehen, wird in diesem Kapitel aufgezeigt, wie diese aufgebaut sind, implementiert und manipuliert werden.

Die Entwicklungsumgebung von Arduino bringt eine Anzahl an Bibliotheken mit sich und erweitert somit die Funktionalität der Arduino-Umgebung. In der Arduino-Software-Distribution finden sich direkt nach der Installation von Arduino integrierte Bibliotheken, welche die gängigen Aufgaben übernehmen. Somit stehen dem Entwickler weitere Befehle und Fähigkeiten zur Verfügung, die der Arduino-Kern vorerst nicht unterstützt. Des Weiteren werden von Community-Mitgliedern Bibliotheken geschrieben, um eine einfache Programmieroberfläche für spezielle Bauteile bereitzustellen. Oftmals werden von Hardwareherstellern ebenfalls Bibliotheken publiziert, um zum Beispiel dem Benutzer eine einfache Anbindung von Sensoren an das Arduino zu gewährleisten und einen großen Funktionsumfang bereitzustellen.

Unter anderem können Bibliotheken auch einen komplexen Code kapseln, um somit eine strukturiertere und einfachere Verwendung zu ermöglichen. So zum Beispiel die von Arduino mitgelieferte Wire-Bibliothek, welche die grundlegende Hardwarekommunikation managet.

Im Arduino-Verzeichnis »hardware/libraries« befinden sich die gespeicherten Bibliotheken, welche dem Entwickler zur Verfügung stehen. Hierbei handelt es sich um Sammlungen von C und C++ Quellcodedateien, die in diesem Verzeichnis abgelegt werden. Pro Bibliothek existiert eine Header-Datei, welche weder die Funktionalität noch den eigentlichen Quellcode enthält, sondern die Oberfläche beschreibt, die der Entwickler in seinem Programm aufrufen kann. Werden Bibliotheken aus dem Internet geladen, so müssen diese in dem oben erwähnten Verzeichnis abgelegt werden, damit die Arduino-Umgebung beim Start die dementsprechende Bibliothek indizieren kann und sie dem Anwender zur Verfügung steht. Die Headerdatei wird über eine Zeile, `#include <headerdatei.h>`, in den Sketch eingebunden. Die Entwicklungsumgebung erkennt diese Zeile und sucht im Verzeichnis nach der dementsprechenden Datei, welche beim Hochladen auf das Board mit dem restlichen Programmcode übersetzt wird. [5], [9]

2.6.1 Standard-Bibliotheken

Die Standard-Bibliotheken werden mit der Installation der Entwicklungsumgebung automatisch implementiert. Jede dieser Bibliotheken enthält Beispiel-Sketche, welche die Verwendung demonstrieren und eine schnelle Anwendung der jeweiligen Bibliothek ermöglicht. Momentan werden 13 dieser Bibliotheken mit installiert.

- **EEPROM**
  Zum Schreiben und Lesen von Daten in und aus dem gleichnamigen Speicher (siehe Kapitel 2.2.2 »Atmel-Mikrocontroller«).
- **Ethernet**
  Für die Kommunikation mit dem Internet unter Verwendung des Arduino Ethernet-Shields.
- **Firmdata**
  Ist ein Protokoll, welches die serielle Kommunikation und Steuerung des Boards vereinfacht.
- **GSM**
  Für die Verbindung zu einem GSM[37]-/GPRS-Netzwerks mit dem GSM-Shield.
- **LiquidCrystal**
  Für die Steuerung von kompatiblen Liquid-Crystal-Displays (LCDs).
- **SD**
  Zum Lesen und Beschreiben von SD-Karten.
- **Servo**
  Zur Steuerung von Servo-Motoren.
- **SPI**
  Zur Kommunikation mit Geräten, welche den SPI-Bus verwenden.
- **SoftwareSerial**
  Ermöglicht an jedem digitalen Pin eine serielle Kommunikation.

[37] Akronym für »Global System for Mobile Connections« , ein Standard für volldigitale Mobilfunknetze

- **Stepper**
  Zur Steuerung von Schrittmotoren.
- **TFT**
  Zum Darstellen von Text, Bildern und Grafiken auf dem Arduino TFT-Bildschirm.
- **WiFi**
  Für die Kommunikation mit dem Internet unter Verwendung des Arduino WiFi-Shields.
- **Wire**
  Zum Senden und Empfangen von Daten über »Two-Wire-Interface« (TWI/I2C).

Die Standard-Bibliotheken unterscheiden sich bei den verschiedenen Boards, diesbezüglich ist bei einem Arduino-Robot auch eine »Robot«-Bibliothek vorhanden, um die spezifischen Funktionen dieses Boards nutzen zu können. Des Weiteren wird auf der Arduino-Referenzseite eine Vielzahl an Erweiterungs-Bibliotheken bereitgestellt (*http://arduino.cc/en/Reference/Libraries*). [50]

Durch den Arduino-Build-Prozess wird beim Kompilieren nur der Teil der Bibliothek in den Programmcode übernommen, der auch wirklich benötigt wird. Somit fallen keine unnötigen Daten an, die den begrenzten Speicher verbrauchen könnten.

2.6.2 Bibliotheken von Drittanbietern

Für viele Hardwarekomponenten existieren zugehörige Bibliotheken. Diese kommen entweder vom Hersteller selbst und sind dann meist gut dokumentiert oder werden von sonstigen Drittanbietern dafür geschrieben. Diese Bibliotheken ermöglichen eine schnelle Inbetriebnahme von Geräten, Sensoren oder Aktuatoren. Die Dateien müssen hierzu nur heruntergeladen, gegebenenfalls entpackt, in das Bibliotheks-Verzeichnis gelegt und anschließend die Arduino-Software neu gestartet werden, damit die Bibliothek indiziert und verwendet werden kann. In Kapitel 2.1 »Projekte« wurden bei einer Anwendung Dallas-Temperatur-Sensoren verwendet. Für diese sind Bibliotheken erhältlich, welche Funktionen anbieten, um die Werte einfach und schnell in der korrekten Einheit (Celsius oder Fahrenheit) auslesen zu können.

2.6.3 Bibliotheken manipulieren

Entwickler von Bibliotheken legen diese meist so an, dass übliche Szenarien abgedeckt werden. Allerdings kommt es auch vor, dass eine Funktion nicht den eigenen Anforderungen entsprechend arbeitet und dahingehend muss diese dem Anforderungsprofil angepasst werden.

Hierzu muss die Bibliothek installiert sein. Dies umschließt ebenfalls die Indizierung durch die Arduino-Software. Anschließend kann die Header-Datei in dem Verzeichnis der jeweiligen Bibliothek mit einem Text-Editor geöffnet werden. Hier werden alle weiteren Dateien inkludiert und erste Variablen deklariert. Je nach Aufbau der Bibliothek und Änderungswunsch muss nach der entsprechenden Position oder Funktion im Programmcode gesucht werden und kann anschließend verändert werden. Sollte eine Bibliothek nach Bearbeitung nicht mehr funktionsfähig sein, so kann diese gelöscht und noch mal heruntergeladen und somit der Originalzustand wieder hergestellt werden. [5]

2.6.4 Verwendung der Ethernet-Bibliothek

Diese Bibliothek wird in Verbindung mit einem Ethernet-Shield verwendet und ermöglicht es, mit einem Server im Internet zu kommunizieren. Trotz der schlichten Bezeichner wie »Ethernet« und »WiFi« geht der Funktionalitätsumfang der beiden Bibliotheken weit über den Umfang des jeweiligen Protokolls hinaus. Diese implementierten einerseits Server-Klassen, welche es ermöglichen verschiedene Serverdienste, wie zum Beispiel Web-Server oder Telnet-Server, dem Arduino bereitzustellen. Andererseits bietet sie auch Client-Klassen, durch die der Arduino mit Servern kommunizieren kann. Des Weiteren umfassen diese den TCP/IP-Stack und das UDP-Protokoll. Um eine der beiden Funktionen (Client/Server) der Bibliothek verwenden zu können, muss die Bibliothek initialisiert werden.

```
//Initialisierung über Mac-Adresse und IP
Ethernet.begin(mac,ip);

//Initialisierung über Mac-Adresse, IP und Gateway
Ethernet.begin(mac,ip,gateway);

//Initialisierung über Mac-Adresse, IP, Gateway und Subnetz
Ethernet.begin(mac,ip,gateway,subnet);
```

Quellcode 4: Ethernet initialisieren

Mit einer dieser Funktionen (siehe Quellcode 4) wird der Ethernet-Chip auf dem Shield initialisiert und die Ethernet-Adresse und IP-Adresse des Arduino festgelegt. Diese Funktion muss vor der Verwendung von Server- oder Client-Funktionen ausgeführt werden. Die Angabe von Gateway und Netzwerkmaske ist optional. [5], [9]

Die Implementierung in den Sketch sieht demnach wie folgt aus.

```
#include <Ethernet.h>
byte mac[] = {0xde,0xad,0xef,0xfe,0xed};
byte ip[] = {10,0,0,27};

void setup() {
    Ethernet.begin(mac,ip);
}

void loop() {
}
```

Quellcode 5: Beispiel Implementierung – Ethernet-Bibliothek [9]

Server-Funktionen

Durch die Serverfunktionalität der Bibliothek ist der Arduino fähig einen Internetdienst anzubieten. Hierzu muss ein Server-Objekt erstellt werden, um Verbindungen annehmen und Daten von Clients lesen und schreiben zu können.

- **Server(Port)**
  Hierdurch wird ein Serverobjekt erzeugt, welches auf den angegeben Port (Telnet-Dienst, Mail-Server, Web-Server, usw.) lauscht.
- **Server.begin()**
  Nach dem Aufrufen dieser Funktion kann das Server-Objekt eine Verbindung annehmen.
- **Server.Verfügbar unter()**
  Hiermit wird überprüft, ob ein Server-Objekt eine Verbindung akzeptiert und angenommen hat. Sollte dies der Fall sein, wird ein Client-Objekt zurückgegeben, welches verwendet werden kann, um mit dem anderen Teilnehmer der Verbindung zu kommunizieren. Wenn nicht, liefert die Funktion »NULL« zurück.
- **Server.write(data)**
  Durch diese Funktion können Daten an alle mit dem Server-Objekt verbundenen Teilnehmer gesandt werden.
- **Server.print(value)**
  Ähnlich der `Serial.print()`-Funktion, können hier formatierte Werte an die verbundenen Teilnehmer übermittelt werden. [5], [9]

Client-Funktionen

Anders als bei der Serverfunktionalität verbindet sich der Arduino hierbei mit einem Internetdienst. Allerdings werden Client-Funktionen ebenfalls verwendet, wenn auf dem Arduino ein Server läuft und ein externer Teilnehmer sich mit diesem verbindet. Dahingehend kann die Clientfunktionalität auch als Verbindungsfunktionalität bezeichnet werden, da diese in beiden Anwendungsfällen auftauchen. Wird Arduino als Server verwendet, so wird bei dem Aufruf von `Server.Verfügbar unter()` ein Client-Objekt zurückgegeben. Arduino als Client verwendet hingegen das Client-Objekt, welches manuell erzeugt werden muss, um eine Verbindung mit einem externen Server herzustellen. Hierzu müssen die Adresse und der Port des Dienstes auf dem dementsprechenden Server angegeben werden. [5]

Über den Aufruf `Client.connect()` wird eine Verbindung zur Gegenseite aufgebaut. Nach erfolgreicher Konnektivität werden mithilfe des Client-Objekts, Daten zwischen Arduino und externem Teilnehmer ausgetauscht.

- **Client.connected()**
  Diese Funktion ist für die Kontrolle einer Verbindung zuständig. Sie gibt an, ob ein benutztes Client-Objekt noch mit der Gegenseite verbunden ist. Daher liefert diese `TRUE` zurück, sollte noch eine TCP[38]-IP Verbindung stehen und `FALSE`, falls nicht mehr.
- **Client.connect()**
  Durch diese Funktion versucht sich das Ethernet-Shield mit der Gegenseite zu verbinden. Gelingt dies, wird `TRUE` zurückgegeben und im anderen Fall `FALSE`.
- **Client.Verfügbar unter()**
  Bei einer TCP-Verbindung werden die empfangenen Daten auf dem Arduino gepuffert. Durch diese Funktion lässt sich abfragen, ob sich Daten in diesem Puffer befinden, welche anschließend über `Client.read()` ausgelesen werden können.

[38] Akronym für »Transmission Control Protocol«, zu deutsch »Übertragungssteuerungsprotokoll«

- **Client.read()**
  Durch die Prüfung von `Client.Verfügbar unter()` weiß man über die Datenmenge der empfangenen Daten Bescheid. `Client.read()` kann einzelne Bytes aus dem Puffer auslesen. Sollten allerdings keine vorhanden sein, so liefert diese Funktion den Wert »-1« zurück.
- **Client.flush()**
  Der Puffer der Verbindung wird geleert, somit sind alle nicht eingelesenen Bytes gelöscht.
- **Client.stop()**
  Durch diese Funktion wird die TCP-Verbindung zur Gegenseite getrennt.
- **Client.write(data)**
  Mit dieser Funktion können Daten an alle, mit dem Client-Objekt verbundenen Teilnehmer, geschrieben werden.
- **Client.print(value)**
  Ähnlich wie bei `Server.print(value)` können hiermit Daten gesandt werden, allerdings nur an die Gegenstelle der Verbindung. [5], [9]

Die WiFi-Bibliothek ist ähnlich wie die Ethernet-Bibliothek aufgebaut und deren Funktionen wie auch Anwendung wird jeweils über Beispiel-Sketche von der Arduino-Entwicklungsumgebung erläutert, um somit einen schnellen Einstieg in den Umgang mit der jeweiligen Standard-Bibliothek zu erlangen. Es können auch eigene Bibliotheken erstellt werden, um somit den eigenen Code übersichtlich zu halten und oft verwendete Funktionen auszulagern. Eine genaue Erklärung hierzu und über weitere Standard-Bibliotheken ist unter *http://arduino.cc/en/Reference/Libraries* vorzufinden.

2.7 Hardwareerweiterungen (Shields)

Um den Funktionalitätsumfang eines Arduinos zu erweitern oder umfangreichere Projekte zu realisieren, werden Hardwareerweiterungen benötigt. Diese basieren meist auf Zusatzplatinen mit integrierten Schaltungen. Durch diese Erweiterungen lassen sich zum Beispiel weitere Sensoren oder Aktuatoren anschließen oder das Arduino mit dem Internet verbinden. Mittlerweile bietet die sehr umfangreiche Produktpalette von Arduino eine Vielzahl an Boards für unterschiedliche Anwendungsfälle an, allerdings ist man oftmals dennoch gezwungen das Board aufzurüsten, um alle Hardwarekomponenten oder Funktionen für ein Projekt in Betrieb zu nehmen. Aus finanzieller Sicht ist es zudem teilweise profitabler ein Standard-Arduino-Board mit einem Shield eines Drittanbieters aufzustocken, als das dementsprechende Board von Arduino zu beziehen.

Einige Shields verfügen über, für das entsprechende Board, kompatible Stiftleisten und können daher direkt auf das Arduino-Board aufgesteckt werden, andernfalls müssen die Shields über Drahtverbindungen mit den jeweiligen Pins am Arduino-Board verbunden werden. Auf den Erweiterungsplatinen sind die Zusatzschaltungen und Anschlusskomponenten untergebracht und können durch die von der Entwicklungsumgebung implementierten oder von Drittanbietern bereitgestellten Bibliotheken angesprochen werden.

Anders, als die eben genannten Erweiterungsplatinen, ist das »Protoshield«, welches dem Anwender universelle Erweiterungen ermöglicht. Wie der Name schon besagt, handelt es sich hierbei um ein Prototype-Shield. Dieses wird auf das Arduino-Board gesteckt und

bietet dem Entwickler Lötpunkte auf einer Leiterplatte an, um das Shield individuell zu bestücken. Aufgrund dessen wird keine Erweiterung von Funktionen für den Arduino selbst herbeigeführt, sondern eine erweiterte Benutzerfreundlichkeit. Diese Shields verfügen meist über einen ICSP-Stecker, Reset-Schalter und zahlreiche Anschlussmöglichkeiten für 5 Volt und Masse. Geeignet sind sie für die Entwicklungsphase und um stabile Testschaltungen zu realisieren. [8], [9], [51], [52]

In den folgenden Unterkapiteln werden wichtige Shields und Module für die verschiedenen Netzwerk-Standards wie Bluetooth, (Wireless)-LAN und ZigBee genannt und erläutert. Diese umfassen ausschließlich ein kleines Spektrum der, auf dem Markt erhältlichen Shields für den dementsprechenden Anwendungsfall.

2.7.1 Drahtloskommunikation

Durch Drahtlosmodule für das Arduino-Board wird der Funktionalitätsumfang dahingehend verändert, dass eine Kommunikation mit dem Board auch aus der Ferne möglich ist, ohne auf Drähte oder aufwändige TCP/IP-Verbindungen zurückgreifen zu müssen. Arduino bietet für diesen Bereich auch ein eigenes Board namens »Arduino-BT« an, welches über ein Bluetooth-Modul verfügt und eine kabellose, serielle Kommunikation ermöglicht. Der »Arduino-Fio« wurde ebenfalls für den kabellosen Einsatz entwickelt und verfügt über einen XBee-Anschluss und ermöglicht, XBee-Module für eine Drahtloskommunikation mit diesem zu verbinden. [9], [53], [54]

ZigBee ist, wie im Kapitel 2.1.2 »Hausautomation« erwähnt, ein Standard im Bereich der Drahtloskommunikation. Dies ist auf die einfache Verwendung des Protokolls und auf die hohe Reichweite wie auch dem Niedrigenergie-Standard zurückzuführen. Ursprünglich wurde ZigBee zur Steuerung von Haushaltsgeräten und zur Übermittlung von Sensordaten entwickelt. Durch die Verfügbarkeit von XBee-Shields wurde das Protokoll den Arduino-Boards zugänglich gemacht und findet nun auch Verwendung in Projekten zur Steuerung von Robotern oder Fluggeräten. [9], [55]

Neben Bluetooth und dem ZigBee-Standard ist eine Drahtloskommunikation auch über Funkfrequenzen möglich. Diese Technik der kabellosen Kommunikation wird auch RFID[39] genannt und findet immer mehr Einzug in verschiedenste Bereiche. Hierbei sendet ein Lesegerät elektromagnetische Frequenzen aus und ein stromloser Schaltkreis mit einem Speicher, auf dem Identifikationsdaten gespeichert sind, wird daraufhin angeregt diese Daten zu senden, um sich zu identifizieren. Während der Kommunikation genügt die Hochfrequenzenergie, um den Schaltkreis mit Strom zu versorgen. Über diese Technik lassen sich einige Kilobytes an Informationen über mehrere Meter übermitteln. Eine weitere Variante zur drahtlosen Kommunikation kann über ein GSM[40]/GPS[41]-Shield erfolgen. Allerdings muss dieses mit einer SIM-Karte bestückt werden, wodurch wiederum Kosten entstehen. Auf diese Art von Shield wird im Folgenden nicht näher eingegangen. [9], [56], [57]

[39] Akronym für »Radio Frequency Identification«, zu deutsch die Identifizierung mit Hilfe elektromagnetischer Wellen

[40] Akronym für »Global System for Mobile Communication«, ist ein Standard für globale Systeme im Bereich der mobilen Kommunikaiton

[41] Akronym für »Global Positioning System«, zu deutsch das Globale-Positionsbestimmungssystem

Arduino Wireless-SD-Shield

Dieses Wireless-SD-Shield ermöglicht einem Arduino die drahtlose Kommunikation und verfügt zusätzlich über einen Micro-SD Steckplatz. Ermöglicht wird eine derartige Kommunikation über ein ZigBee-XBee- oder gleichartiges Modul, welches auf dem Shield über Stiftleisten angebracht wird. Das ZigBee-XBee-Modul erlaubt eine serielle Kommunikation in Gebäuden über eine Distanz von 100 m und im Freien bis zu 300 m. Mit diesem Shield kann zum Beispiel eine einfache drahtlose Kommunikation zwischen zwei Arduino-Boards oder anderen Geräten, die den ZigBee-Standard verwenden, hergestellt werden. Hierzu wird ein Shield mit Board als Sender, das andere Modul als Empfänger, wie in der folgenden Abbildung, eingerichtet. Über einen Schalter auf dem Shield kann zwischen einer Kommunikation vom Shield zum Mikrocontroller oder zum USB-zu-Seriell-Wandler gewählt werden. [58], [59], [60], [61]

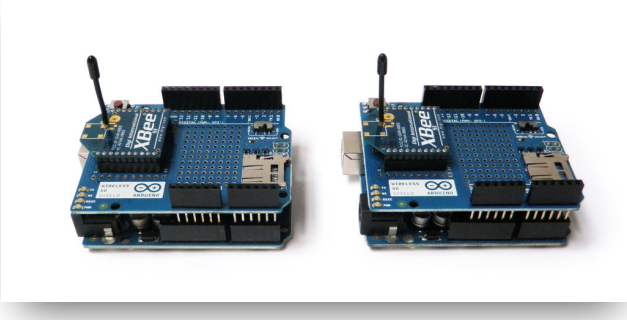


Abbildung 24: Arduino Wireless-SD-Shield [59]

Adafruit PN532 NFC/RFID-Shield

Das Adafruit-PN532-Shield ist eine Hardwareerweiterung für 13,56 MHz- oder NFC[42]-Anwendungen. Der dabei eingesetzte RFID-Transponder ist meist in Form von Etiketten oder Karten vorzufinden und verfügt gegenüber einem 125 kHz-Transporter über eine wesentlich höhere Datenübertragungsrate und ist zudem preisgünstiger. Dieser wird für Leseabstände von bis zu 70 cm eingesetzt. Der auf dem Shield angebrachte NFC-Chip ist der auf dem Markt beliebteste und meist verwendete seiner Art. Anders als bei Bluetooth oder WiFi, ist NFC einfach zu konfigurieren und bedarf keiner vorausgehenden Paarung der beiden in Verbindung tretenden Geräte. Der Kartenleser ist hierbei aktiv, was das Aufbauen eines RF-Feldes für die Datenübertragung bedeutet. Der NFC-Tag ist ähnlich wie eine Spule gewickelt, worüber eine Induktionsspannung aufgebaut und kommuniziert werden kann. Deshalb ist bei den Tags keine externe Stromquelle nötig. Bei einer Datenübertragungsrate von nur 848 Kbps ist dieser wesentlich langsamer als Bluetooth oder WLAN und bei der nur sehr kurzen möglichen Distanz zwischen Leser und Tag von maximal 10 cm auch schlechter als RFID, allerdings besticht dieser über seine weite Verbreitung und den reibungslosen Verbindungsaufbau. [62], [63]–[65]

Das Shield von Adafruit kann I2C- und SPI-Kommunikationsprotokolle verwenden und ist zu den Standard-Boards von Arduino kompatibel in Bezug auf die Stiftleisten.

[42] Akronym für »Near Field Communication«, zu deutsch »Nahfeldkommunikation«, ein internationaler Übertragungsstandard

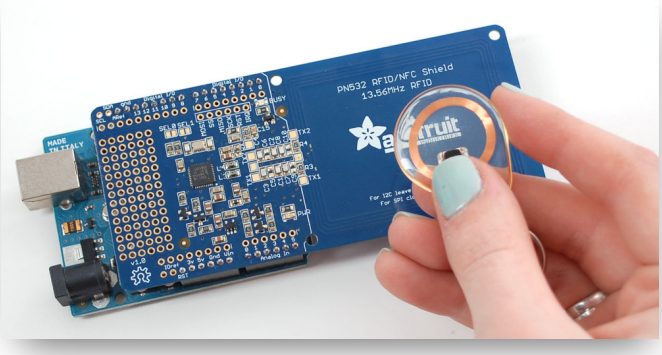


Abbildung 25: Adafruit PN532 NFC/RFID-Controller-Schild [62]

Bluetooth 4.0 Low Energy – BLE-Shield v2.0

Dieses Shield ermöglicht eine Kommunikation über den Industriestandard Bluetooth. Die Übertragung zwischen den Geräten findet hierbei über Funktechnik und nur kurzen Distanzen bis zu 10 m statt. Neben einer internen Antenne verfügt dieses Shield ebenfalls über einen Anschluss, um externe Erweiterungsantennen anzuschließen. Durch die BLE[43]-Protokollarchitektur wird der Energieverbrauch wesentlich gesenkt und eine kostengünstige Integration ist möglich. Dieses Shield ermöglicht die Steuerung von Arduino-Pins durch die vom Hersteller entwickelte oder eigenes programmierte Applikation für Smartphones. Des Weiteren können auch Sensordaten vom Board beziehungsweise Shield an die dementsprechende Applikation übermittelt werden. Diese Hardwareerweiterung ist für die Arduino-Standard-Boards konzipiert und durch einen SPI-Anschluss für verschiedene weitere Shields ausgelegt. [66]–[68]

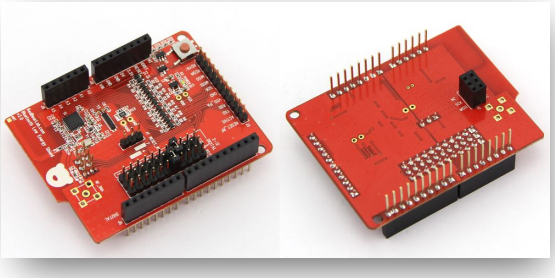


Abbildung 26: BLE Shield v2.0 RedBearLab [66]

Fazit

Auf GSM/GPS- und WiFi-Shields wurde nicht näher eingegangen, da diese preislich weit über den günstigen Lösungen liegen, die in diesem Bereich von Drittanbietern zu beziehen sind. Eine Verwendung muss dahingehend wohl überlegt sein in Bezug auf das zu übermittelnde Datenvolumen, Reichweite und Preis.

RFID-Module sind zwar sehr günstig, allerdings ist bei ihnen auch die Leistung sehr begrenzt. XBee sticht bei den anderen Bereichen hervor, da diese nicht nur für den ZigBee-Standard konzipiert sind, sondern auch als Bluetooth-Module oder für den IEEE

[43] Akronym für »Bluetooth Low Energy«

802.15.4-Standard zu erhalten sind und der Sockel dieser Module weit verbreitet ist. XBee bietet für Arduino die meisten Möglichkeiten und ein sehr gutes Preis-Leistungs-Verhältnis.

Im folgenden Kapitel wird auf Ethernet und Netzwerkkommunikation eingegangen. Hierbei kommen ebenfalls Shields zum Einsatz, die teilweise auch eine drahtlose Kommunikation ermöglichen, allerdings handelt es sich hierbei um komplexere Systeme mit höheren Anforderungen, weshalb sie in diesem Kapitel nicht erwähnt wurden.

2.7.2 Netzwerkkommunikation

In dem Kapitel über Drahtloskommunikation wurden Shields vorgestellt, welche hauptsächlich eine Kommunikation zwischen zwei Geräten ermöglicht. Über Ethernet/WLAN und Netzwerke hingegen lässt sich eine breite Masse an Teilnehmern bewerkstelligen. Diese Module und Shields sind in der Konfiguration und beim Verbindungsaufbau wesentlich komplexer, bieten allerdings die Möglichkeit, Internetdienste zu nutzen und wesentlich höhere Datenraten bei der Übermittlung zu erzielen. Hierbei wird in zwei Kategorien unterschieden. Einerseits ist eine Kommunikation über ein kabelgebundenes LAN[44]-Netzwerk möglich, andererseits lassen sich auch über Funk, und somit kabellos, sogenannte WLAN-Netzwerke aufbauen. Dahingehend unterscheiden sich die Shields für diese Art der Kommunikation in eben diese Bereiche.

Arduino WiFi Shield

Das Arduino WiFi-Shield ermöglicht eine kabellose Verbindung ins Internet. Hierbei wird die 802.11-Spezifikation (unterstützt wird 802.11b/g) zur Kommunikation genutzt. Der auf dem Board sich befindende ATmega-Mikrocontroller bietet ein Netzwerk-IP-Stack[45], welcher sowohl TCP wie auch UDP unterstützt. Ein Micro-SD-Katenslot ermöglicht zudem das Speichern von Webseiten oder das Loggen von Daten. Für die WiFi- und SD-Funktionalität stehen dem Arduino-Shield Bibliotheken zur Verfügung und es kann eine Verbindung mit verschlüsselten oder offenen Netzwerken hergestellt werden. Die Firmware wird über einen Mini-USB-Anschluss erneuert oder angepasst. Dieses Shield ist kompatibel mit dem Arduino-UNO wie auch -Mega. [69]

Abbildung : Arduino WiFi-Shield [70]

[44] Akronym für »Local Area Network«, zu deutsch »lokales Netzwerk«

[45] auch Netzwerkstapel genannt, bezeichnet in der Datenübertragung die Architektur von Kommunikationsprotokollen

Sparkfun WiFly-Shield

Von den Funktionen und der Netzwerkfähigkeit ist dieses Shield ähnlich dem Arduino WiFi-Shield. Allerdings ist die Kommunikation zwischen Shield und Board eine andere. Während das Arduino WiFi-Shield per SPI mit dem Board verbunden wird, geschieht dies beim Sparkfun WiFly-Shield per UART. Auch für das WiFly-Shield existieren zur Anbindung verschiedene Bibliotheken, um eine Verbindung ins Internet herzustellen. Eine Verbindung zu WEP-Netzwerken ist standardgemäß nicht vorgesehen, kann aber durch Bibliotheken von Drittanbietern ergänzt werden. Dieses Shield verfügt über keinen SD-Kartenslot, ist allerdings etwas günstiger als das Arduino eigene WiFi-Shield. [25], [71], [72]

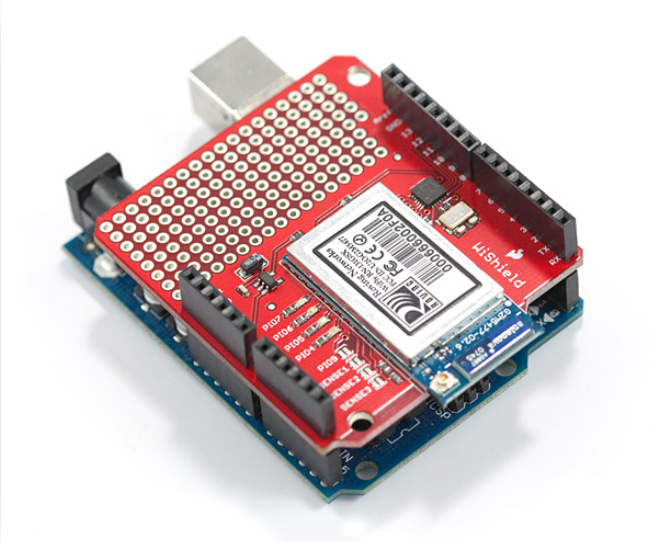


Abbildung 27: Sparkfun WiFly Shield [71]

Arduino Ethernet-Shield

Dieses Shield ermöglicht eine schnelle Anbindung des Arduino-Boards an das Internet. Wie das hauseigene WiFi-Shield unterstützt auch dieses die TCP- und UDP-Netzwerkprotokolle. Ebenfalls verfügt dieses über einen SD-Kartenslot und kann durch eine proprietäre Hardwareerweiterung über ein Standard-Ethernet-Kabel (Kategorie 5) mit Strom versorgt werden. Die Kommunikation findet auch bei diesem Shield über den SPI-Bus statt. [73]

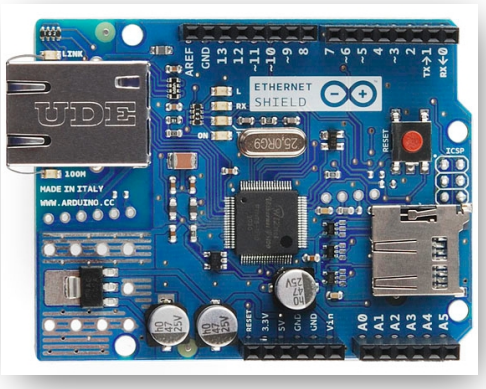


Abbildung 28: Arduino Ethernet Shield [73]

Arduino Yun

Der neuartige Arduino-Yun ist keine reine Erweiterungshardware, sondern vereint auf einem Board einen Arduino-Leonardo ähnlichen Prozessor mit einem WiFi-Chip. Das Board verfügt über ein Ethernet-, WLAN- und SD-Karten-Modul, wobei die Verarbeitung der Netzwerkdaten auf dem vorinstallierten Linux-System (Linino) ablaufen. Durch diese Erweiterungen auf einem Board wird das Arduino als Komponente entlastet und es sind komplexere Web-Anwendungen und Interaktionen möglich. Durch die Linux-Distribution wird ein leistungsstarker Netzwerkcomputer bereitgestellt, vereint mit der leichten Bedienbarkeit eines Arduino. Die von Arduino bereitgestellten Bibliotheken erleichtern die Kommunikation zwischen dem Arduino-Mikrocontroller und dem Linux-Prozessor. Es ist der erste Arduino, welches über eine derartige Distribution verfügt und dadurch auch fähig ist, drahtlos programmiert zu werden. Zudem ist es eine preiswertere Variante als ein Standard-Arduino-Board gepaart mit einem Erweiterungs-Shield für WLAN- oder Ethernet-Kommunikation. Zudem können durch die Linux-Distribution erstmals bei einem Arduino Linux-Kommandos, eigene Shell- und Python-Skripte verwendet werden. [74]–[76]

Auch wenn der Yun nicht unter der Bezeichnung »Shield« läuft, kann dieser dennoch in diesem Kapitel gelistet werden, da sein Funktionalitätsumfang, mögliche Netzwerkkommunikation und geringer Preis eine oftmals bessere Variante für eine Arduino-Netzwerklösung bietet, als eine Kombination aus Board und Shield.

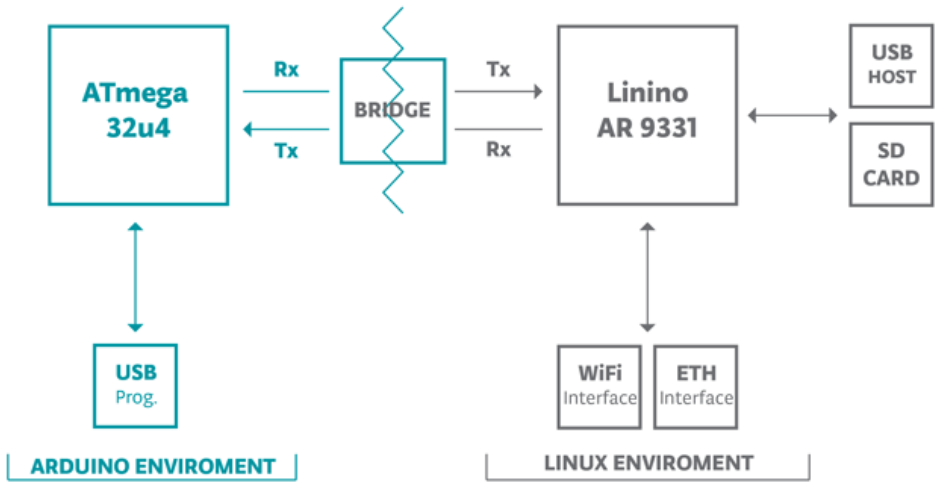


Abbildung 29: Arduino Yun – Systemarchitektur [74]

Fazit

Generell sind Ethernet-Shields wesentlich preiswerter als WiFi-Shields, allerdings ist für einen Betreiber oder Entwickler die Abhängigkeit des Mikrocontroller von einem Netzwerkkabel und somit eine gewisse Einschränkung im Anwendungsumfeld oftmals hinderlich. Der Arduino-Yun hingegen ist preiswert und verfügt über beide Standards und ist somit eine optimale Lösung für viele Arduino-Netzwerkprojekte. Will man ein Arduino hingegen klein halten, wenige Daten nur übertragen und ist auf keine Lniux-Distribution angewiesen, so ist dieses Board meist zu überladen und lässt sich aufgrund des zusätzlichen Prozessors und erweiterten Schnittstellen nicht kompakter auf einem Schaltkreis konzipieren. Ein Vorteil bietet die kabellose Programmierbarkeit dieses Boards, welches dadurch innerhalb eines Netzwerks ohne die Verwendung der USB-Schnittstelle konfiguriert und programmiert werden kann.

2.8 Arduino in Netzwerken

Sensordaten weitläufig teilen, Aktuatoren aus der Ferne steuern oder Informationen aus dem Internet abrufen, all das kann mit einem Arduino bewerkstelligt werden. Die Hardwareanforderungen wurden im letzten Kapitel erläutert. Damit ein Standard-Arduino-Board über diese Funktionen verfügen kann, ist bei den meisten Modellen Erweiterungshardware notwendig, sogenannte Shields.

In Zusammenhang mit Arduino und Internet fällt häufig der Begriff »Internet of things[46]«, welcher den heutigen Einsatz des Internets, losgelöst von dem klassischen Computer als Schnittstelle, darstellen soll. Die Menschheit muss nicht mehr Gegenstand einer solchen Interaktion sein, sondern kann durch »intelligente« Geräte teilweise abgelöst werden, welche die Menschen unmerklich bei ihren Tätigkeiten unterstützt. Eine Verbindung aus physischen Objekten mit dem Internet. Dahingehend sind nicht nur noch Menschen Teilnehmer im Netz, sondern werden durch Geräte ergänzt, die ebenfalls Daten abrufen, Informationen verteilen oder streuen. Arduino kann in diesem Zusammenhang als Client arbeiten und Informationen aus dem Netz abrufen oder fungiert als Sever, welcher Informationen über Internetprotokolle an Clients liefert. [5], [49], [77]

In diesem Kapitel werden grundlegende Eigenschaften und Möglichkeiten zum Verbinden und Austausch zwischen Arduino und dem Internet dargestellt. Die hier verwendete Hardware und Bibliotheken beziehen sich auf eine Ethernet-Verbindung. Eine kabellose Verbindung, über WLAN, ist ebenfalls möglich und in Bezug auf die Bibliotheken zur Verwendung auch nahezu identisch einzusetzen.

Die in diesem Kapitel aufgeführten Anwendungsbeispiele wurden vom Autor dieser Thesis durchgeführt und auf Funktionalität überprüft.

2.8.1 Grundlegende Begriffserklärung

Damit in den folgenden Kapiteln die Verwendung von Akronymen und Konzepten zu keiner Verwirrung führt, werden hier die Wichtigsten erläutert. Allerdings handelt es sich hierbei nur um einen groben Überblick über Netzwerkadressierung und Protokolle.

Ethernet

Ethernet befindet sich auf der untersten Schicht, Signalisierungsschicht, des Netzwerkprotokolls (IP[47]), welche für die physikalische Übertragung zuständig ist. Bei Ethernet handelt es sich um eine kabelgebundene, standardisierte Technologie zur Datenübertragung. Die Quell- und Zieladressen der zu übertragenden Nachricht werden über MAC[48]-Adressen festgelegt. [5], [34]

[46] »Internet of things«, zu deutsch »Internet der Dinge«

[47] Akronym für »Internet Protocol«

[48] Akronym für »Media Access Control«

TCP/IP

Die zwei wichtigsten Protokolle im Internet sind das »Transmission Control Protocol« (TCP) und das Internetprotokoll (IP). Beide sind im TCP/IP-Protokollstapel über der Signalisierungsschicht, auf der sich zum Beispiel Ethernet befindet, angebracht.
TCP kann in etwa mit Übertragungs-Kontroll-Protokoll übersetzt werden und ermöglicht die Übertragung von Informationen über das lokale oder globale Netz und eine verlustfreie Kommunikation. Bei IP handelt es sich um ein Adressierungsprotokoll, welches zur eindeutigen Adressierung der zu übertragenden Daten dient. Somit gelangen diese von einem Sender zu einem bestimmten Empfänger. Die IP-Adressen der Server sind hierbei immer eindeutig und kommen daher nur einmal vor. Dadurch wird gewährleitstet, dass Pakete beim richtigen Empfänger ankommen. Da das Internet das »Domain Name System« (DNS) verwendet, können somit Domainnamen in numerische IP-Adressen gewandelt werden, was auch bei Arduino genutzt werden kann. Lokale IP-Adressen werden in Heimnetzwerken verwendet und diese Adressen werden den Rechnern vom Router zugeteilt. Durch das »Dynamic Host Configuration Protocol« (DHCP) beziehungsweise durch dessen Dienst werden diese Adressen erzeugt. Die Arduino Ethernet- und WiFi-Bibliothek verfügt ebenfalls über diesen Dienst. [2], [33]

MAC-Adresse

Die MAC-Adresse ist eine weltweit eindeutige Adresse, die jedem Netzwerkadapter zugewiesen ist, egal ob Ethernet- oder WiFi-Adapter. Dadurch können physikalische Systeme im Netzwerk ausfindig gemacht werden. [5]

HTTP

Das »Hypertext Transfer Protocol« ist ein Protokoll für die Kommunikation über das weltweite Netz. Web-Requests eines Browsers und der daraus resultierende Response werden über HTTP-Nachrichten versandt. Damit Client und Server mit diesen Nachrichten umgehen und darauf reagieren können, müssen diese das Protokoll verstehen. [34]

HTML

»Hypertext Markup Language« ist eine Auszeichnungssprache, in der üblicherweise Webseiten formatiert sind. [33], [34]

Des Weiteren wurden Web-Austauschformate wie XML[49] und JSON[50] entwickelt um eine zuverlässige Extrahierung von übermittelten Web-Daten per Software zu ermöglichen. [5]

2.8.2 Netzwerkkommunikation

Wie zuvor erwähnt, kann die Kommunikation zwischen den Komponenten eines Systems auf dem »Transmission Control Protocol« basieren. Eine derartige Verbindung kann in zwei Bereiche gegliedert werden. Zum einen gibt es den lokalen Bereich, in dem durch Ethernet oder WLAN eine Verbindung zum Router hergestellt werden muss, zum anderen muss auf globaler Ebene der Server das lokale Netzwerk mit dem Internet verbinden.

[49] Akronym für »Extensible Markup Language«
[50] Akronym für »JavaScript Object Notation«, ein kompaktes Datenformat

Internet-Sockets[51] sind für die Kommunikation mittels Kommunikationsprotokolle zuständig. Dabei wird in Stream- und Datagramm-Sockets unterschieden. Für gewöhnlich verwenden Stream-Sockets das zuverlässige TCP-Protokoll und Datagramm-Sockets das unzuverlässige, dafür schnellere »User Datagramm Protocol« (UDP). Die Teilnehmer einer Verbindung kommunizieren über das Infrastruktur-Prinzip mit dem Server. Hierbei werden die Socket-Verbindungen genutzt. [49], [78]

Das Standardkonzept für die Verteilung von Aufgaben innerhalb eines Netzwerkes wird Client-Server-Modell genannt. Unter Aufgaben kann hierbei zum Beispiel E-Mail-Versand/Empfang, Web-Zugriff oder andere spezifischere Aufgaben verstanden werden. Bei Client wie auch Server handelt es sich um Programme oder Prozesse, welche miteinander kommunizieren. Es können auch beide Programme auf einem System umgesetzt werden, doch im weiteren Verlauf der Thesis wird als Client und Server ein jeweils eigenes System bezeichnet. Eine Client-Server-Kommunikation bei einer Web-Anfrage sieht wie folgt aus. [79]

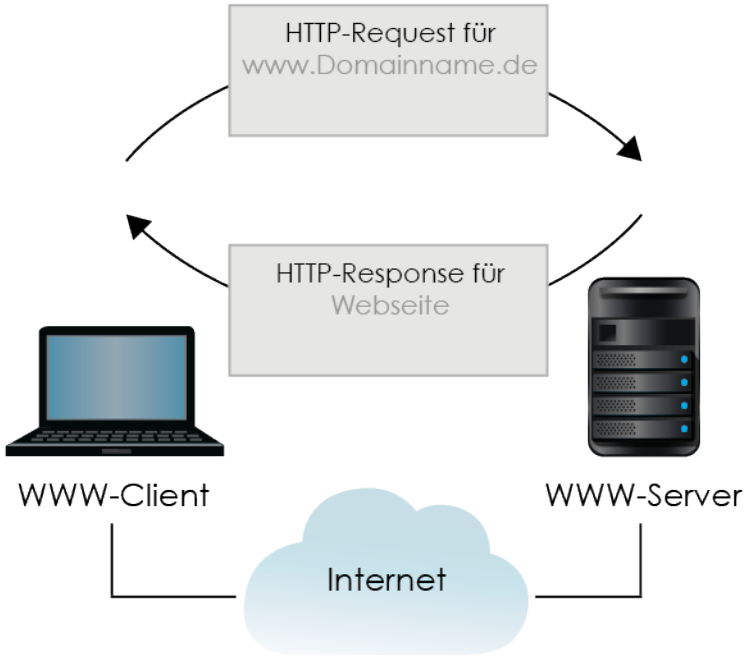


Abbildung 30: Client/Server-Kommunikation

Der Client ist hierbei der Auftraggeber und sendet eine Anfrage an den Server. Dieser ist der Auftragnehmer, bearbeitet die Anfrage des Clients und liefert Informationen zum Beispiel in Form einer Webseite, Datenbankeinträge oder anderen angeforderten Daten zurück. Ein Arduino-Board kann als Client wie auch als Server eingesetzt werden. Beispiele zur Implementierung und Anwendung finden sich in den folgenden Kapiteln.

[51] plattformunabhängige, standardisierte Schnittstelle zwischen Netzwerkprotokoll-Implementierung und Anwendungssoftware

2.8.3 Ethernet-Shield einrichten

Das Ethernet-Shield wird nahezu gleich konfiguriert wie das WiFi-Shield von Arduino. Aufgrund der gleichartigen Konfiguration der beiden Shield-Varianten werden die Erklärungen und Beispiele auf das Ethernet-Shield bezogen und bei Abweichungen gegenüber dem WiFi-Shield werden diese ebenfalls genannt. Das Ethernet-Shield wird in der Praxis häufiger für eine Arduino-Internet-Verbindung verwendet als das WiFi-Shield. Aus diesem Grund ist es sinnvoller sich in dieser Thesis auf eben dieses zu beziehen, gleichgültig welches anschließend in der Entwicklung eines Prototyps verwendet wird.

Die grundlegenden Funktionen, Klassen und Methoden der Ethernet-Bibliothek wurden in Kapitel 2.6.4 »Verwendung der Ethernet Bibliothek« erläutert und sollten so weit bekannt sein. Unterschied zur WiFi-Bibliothek liegt in der verbindungsspezifischen Konfiguration. Anders als bei einer Ethernet-Verbindung, kann bei einer WLAN-Verbindung zwischen verschiedenen Netzwerken, welche in Reichweite liegen, ausgewählt werden. Des Weiteren verfügen die meisten Netzwerke über eine Verschlüsselung, welche ebenfalls berücksichtig werden muss.

Das Shield an sich wird über die Stiftleisten auf dem Arduino-Board aufgesteckt und kann somit über den SPI-Bus mit diesem kommunizieren. Bei einem Arduino entspricht diese Belegung den Pins D10 bis D13, welche dadurch nicht weiterhin zur Verfügung stehen (siehe Abbildung 31). Pin D4 wird im Falle der Verwendung von SD-Karten ebenfalls belegt.

Seit der Veröffentlichung von Arduino 1.0 stehen der Ethernet-Bibliothek einige Verbesserungen zur Verfügung. Zum einen wird die Nutzung von DHCP und DNS vereinfacht und direkt in die Bibliothek implementiert, sodass keine Weiteren hierfür heruntergeladen und eingefügt werden müssen. Zudem wurde die Stream-Parsing-Methode vereinfacht und ein Konstrukt hinzugefügt, welches das Speichern von Text in dem Flash-Speicher unkomplizierter gestaltet. [5], [73]

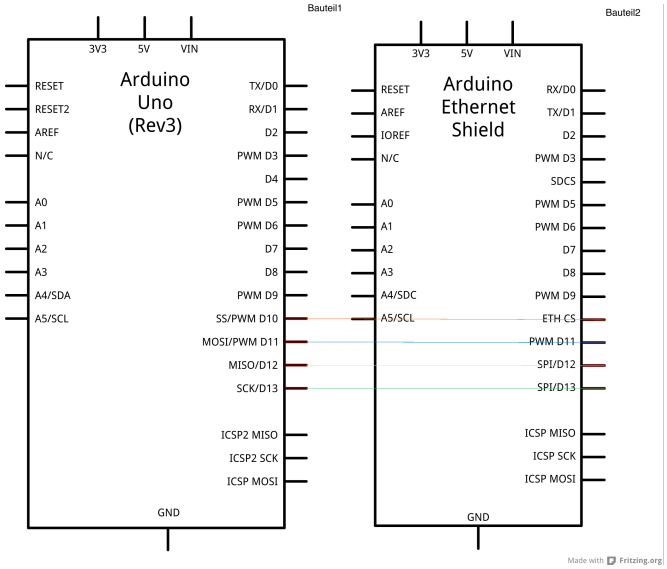


Abbildung 31: Arduino-UNO - Ethernet-Shield Schaltplan

Wenn ein Arduino-Board mit einem Shield verbunden wird, muss in der Regel eine Bibliothek implementiert werden, welche die Kommunikation zwischen den beiden Komponenten managt. Beim Ethernet- wie auch WiFi-Shield handelt es sich um eine SPI-Kommunikation. Daher muss zu Beginn des Sketches und vor dem Ethernet-Include die SPI-Bibliothek inkludiert werden. Seit dem Release von Arduino 1.0.5 ist die Inkludierung der SPI-Bibliothek optional und wird automatisch von der WiFi- und Ethernet-Bibliothek inkludiert.

```
#include <SPI.h>
#include <Ethernet.h>
```

Quellcode 6: Ethernet-Bibliothek inkludieren

Ethernet-Shield mit fester IP-Adresse

In diesem Fall wird eine für das Netzwerk gültige IP-Adresse manuell gesetzt. Diese ist somit statisch und verändert sich auch nach wiederholtem Programmstart nicht.

```
byte mac[] = { 0xDE,0xAD,0xBE,0xEF,0xFE,0xED };
byte ip[] = { 192, 168, 1, 177 };
// Beispiel IP-Adresse von Google
byte server[] = { 209,85,229,104 };
```

Quellcode 7: Ethernet-Shield feste IP-Adresse festlegen

Es müssen bis zu vier Adressen korrekt konfiguriert werden, damit eine Verbindung hergestellt werden kann. Zum einen wird im obigen Quellcode die MAC-Adresse initialisiert. Diese ist auf der Rückseite des Boards angebracht und muss an diese Stelle übertragen werden. Dadurch ist dieses Shield eindeutig identifizierbar. Des Weiteren wird eine für das Netzwerk korrekte, freie IP-Adresse eingegeben. Auch diese muss in dem verwendeten Netzwerk eindeutig sein. Zudem wird in diesem Beispiel-Quellcode die IP-Adresse des Google-Servers als solche angegeben. Falls ein Router die Angabe von DNS und Gateway verlangt, müssen diese ebenfalls noch hinzugefügt werden.

```
byte dns_server[] = { 192, 168, 1, 2 };
byte gateway[] = { 192, 168, 1, 254 };
```

Quellcode 8: DNS und Gateway festlegen

Die eben erwähnten Konfigurationen werden als erstes im Sketch durchgeführt und stehen somit ganz oben. Anschließend wird in der `setup()`-Funktion die Ethernet-Initialisierung durchgeführt. Hierbei werden die Konfigurationen übermittelt.

```
Ethernet.begin(mac, ip, dns_server, gateway);
```

Quellcode 9: Ethernet-Initialisierung

Das WiFi-Shield kann sich mit verschiedenen Netzwerken verbinden. Aus diesem Grund muss über den SSID[52] und mit dem zugehörigen Passwort die Konfiguration vorgenommen werden. Hierbei wird zwischen zwei verschiedenen Verschlüsselungstypen unterschieden. Zum einen existiert der WPA[53]/WPA2-Schlüssel und zum anderen der WEP[54]-Schlüssel, welcher allerdings veraltet und unsicher ist, aber dennoch von Arduino unterstützt wird. Bei der Konfiguration eines WPA/WPA2-Netzwerks werden der SSID und das Passwort benötigt, für eine Verbindung zu einem WEP-Netzwerk sind zusätzlich noch ein Schlüsselindex und der eigentliche Schlüssel erforderlich. [9], [69], [73]

```
WiFi.begin(ssid, pass);
```

Quellcode 10: WiFi-Initialisierung

Ethernet-Shield mit automatischer IP-Adresse

In diesem Fall muss das Ethernet-Shield die IP-Adresse vom DHCP-Server abrufen.

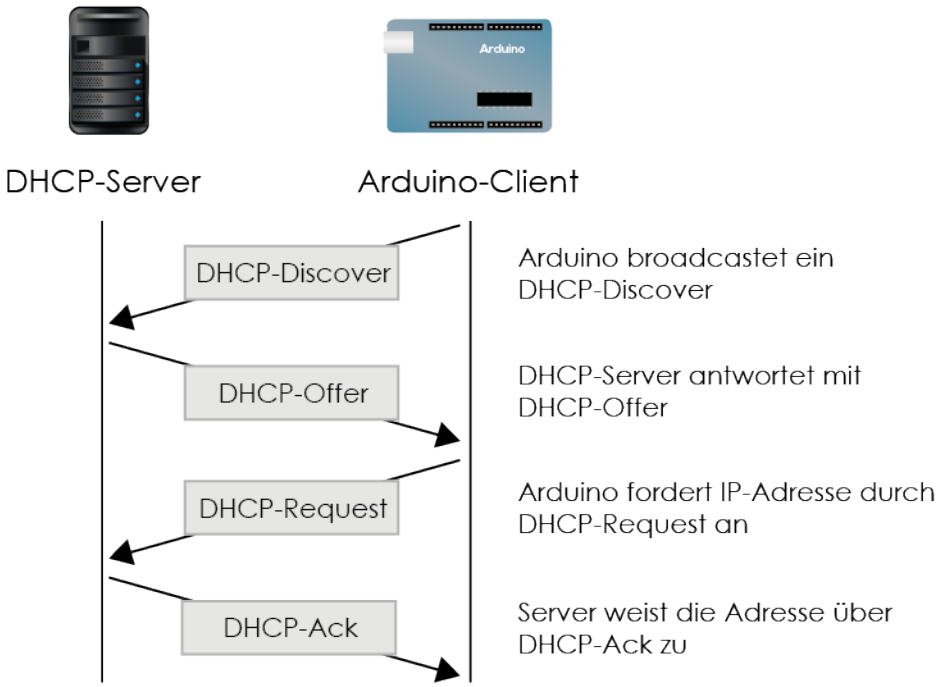


Abbildung 32: DHCP-Ablauf

Beim DHCP-Discover sendet der Arduino als eigene IP-Adresse, da dieser noch keine hat, die »0.0.0.0« und seine MAC-Adresse per Broadcast an alle verfügbaren DHCP-Server. Anschließend senden die DHCP-Server mit DHCP-Offer Vorschläge für eine IP-Adresse, ebenfalls mit einem Broadcast. Mit einem DHCP-Request meldet sich der Arduino beim gewünschten Server mit einem Paket, welches den Serveridentifier enthält. Alle anderen Server beenden daraufhin die Verbindung. Anschließend bestätigt der ausgewählte DHCP-Server noch mittels eines DHCP-Ack die IP-Adresse und nach einem Timeout wird die

[52] Akronym für »Service Set Identifier« und steht für den Namen des Netzwerkes
[53] Akronym für »Wi-Fi Protected Access«
[54] Akronym für »Wired Equivalent Privacy«

Verbindung beendet und der Arduino hat nun eine IP-Adresse bezogen. Die Kommunikation findet hierbei über UDP statt. [80]

Zur Konfiguration genügt es aus, wenn bei der Ethernet-Initialisierung ausschließlich die MAC-Adresse übergeben wird. Der Rest regelt Arduino und der DHCP-Server. Zusätzlich wird in diesem Zusammenhang geprüft, ob der Aufruf von `Ethernet.begin(mac)` erfolgreich war und somit eine IP-Adresse vom DHCP-Server bereitgestellt wurde. Durch eine in Arduino 1.0 eingeführte Klasse namens »IPAdress« ist es möglich ein IPAdress-Objekt über `Serial.print()` auszugeben, um somit die zugewiesene IP-Adresse zu erfahren. [9]

```
//Ethernet mit mac und DHCP starten
if(Ethernet.begin(mac)==0) {
    Serial.print("Verbindung nicht möglich!");
}

//IP-Adresse ausgeben
IPAdress meineIP = Ethernet.localIP();
Serial.print(meineIP);
```

Quellcode 11: IP-Adresse asugeben

DNS nutzen

Statt einer IP-Adresse für den zu kontaktierenden Server kann auch ein Hostname verwendet werden. Die Abbildung des logischen Namens in die entsprechende IP-Adresse verläuft wie folgt.

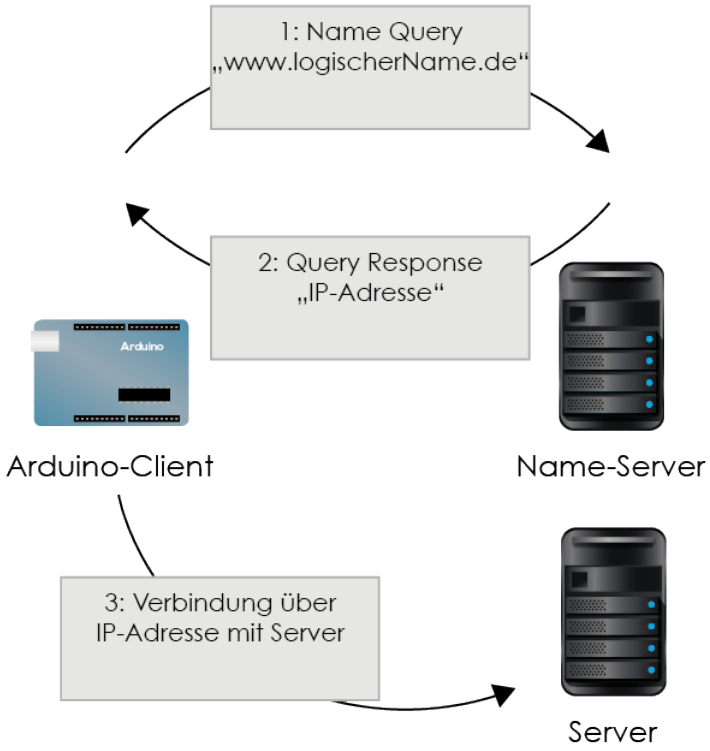


Abbildung 33: Domain-Name-Server

Bei einer Webanfrage über einen logischen Namen wird an einen Name-Server eine Anfrage gestellt (1). Diese verfügen über Datenbanken mit logischen Namen und den zugehörigen IP-Adressen. Wird bei einem dieser Server der Hostname gefunden, bildet er den logischen Namen auf die IP-Adresse ab und liefert diese an den Client zurück (2). Anschließend kann sich der Client mit dem gewünschten Server über die IP-Adresse verbinden (3). [80]

Auf das Verfahren muss bei der Entwicklung von Arduino-Projekten keine Rücksicht genommen werden, da diese Funktionalität mittlerweile fester Bestandteil der Entwicklungsumgebung ist.

```
char serverName[] = "www.google.com";

int ret = client.connect(serverName, 80);
if(ret == 1) {
   Serial.println("Verbunden");
}
```

Quellcode 12: Arduino nutzt DNS

Die `client.connect()`-Funktion liefert den Wert »1« zurück, wenn der DNS-Server den Hostnamen erfolgreich in eine IP-Adresse auflösen konnte.

Weitere Statusmeldungen:

- »0«, die Verbindung ist fehlgeschlagen
- »-1«, es wurde kein DNS-Server angegeben
- »-2«, kein DNS-Eintrag zum Hostname gefunden
- »-3«, Timeout

In dieser Funktion wird zusätzlich noch der Port des jeweiligen Dienstes angegeben. Dadurch ist zum Beispiel der Server fähig zu erkennen, mit welchem Dienst dieser die Anfrage zu verarbeiten hat.

Eine vollständige Web-Anfrage von einem Arduino (Client) an Google (Server) ist in Anhang E einzusehen. Dieses Kapitel dient zur Bereitstellung der Grundlagen und Initialisierung des Boards. In den folgenden Kapiteln wird auf dieser Basis der Arduino als Server und Client praxisbezogen eingesetzt und Möglichkeiten zur Verwendung aufgezeigt.

2.8.4 Arduino als Web-Client

In diesem Kapitel wird der Arduino als Web-Client (siehe Kapitel 2.8.2, Abbildung 30) implementiert und kann verschiedene Funktionen ausführen. Das Anwendungsspektrum ist in diesem Bereich sehr breit gefächert und der Arduino kann als Client von verschiedenen Diensten gebrauch machen. Diese hier beispielhaft erläuterte Anwendung wird Daten von einem Web-Server abrufen. Unter anderem kann der Arduino als Client auch per GET- oder POST-Methode eingelesene Werte von Sensoren als HTTP-Request an Webanwendungen senden. Diese können anschließend serverseitig ausgelesen und zum Beispiel in Skripten (PHP,Pythen, und vielen mehr) verarbeitet werden. Des Weiteren kann

ein Arduino auch Verbindung zu anderen Diensten aufnehmen und komplexere Anfragen realisieren. Diese werden allerdings gesondert in den folgenden Kapiteln erläutert.

In diesem Beispiel werden mittels Arduino als Web-Client bestimmte Daten von einer Suchmaschine angefragt und auf dem Seriellen-Monitor ausgegeben. Hierbei soll eine gewisse Kilometeranzahl in Meilen ausgegeben werden. Die Netzwerkarchitektur entspricht einer einfachen Server/Client-Kommunikation wie in Abbildung 31 aufgezeigt. Als Suchmaschine wird »Yahoo« eingesetzt, da in diesem Fall nach einer definierten Zeichenfolge gesucht wird, welche in deren Anschluss das Ergebnis der Anfrage liefert. »Google« verwendet hingegen bei der gleichen Anfrage eine andere HTML-Struktur und je nach Google-Ranking unterschiedlich formatierte Ergebnisse. Aus diesem Grund ist es schwer konstante Parameter für den Request zu finden.

Die Anfrage beziehungsweise der Request wird per GET-Methode umgesetzt, welche die für die Suche relevanten Parameter beinhaltet. Wie das genau aussieht, wird im weiteren Verlauf mithilfe des Quellcodes genauer erklärt.

In diesem Sketch wird die Stream-Parsing-Funktionalität der Arduino-Entwicklungsumgebung verwendet. Diese Funktion ist sehr hilfreich bei der Extrahierung von Daten aus einer Webseite, da normalerweise ein herkömmliches Display zur Darstellung dient und Bilder, Format-Tags und Texte typischer Weise angezeigt werden. In diesem Fall wird allerdings nur ein gewisser Teil beziehungsweise spezifische Daten verlangt und daher ist die Suche nach eben diesen für gewöhnlich sehr umständlich. Aus diesem Grund bietet Arduino die Stream-Parsing-Funktion an, um bestimmte Zeichenfolgen aufzuspüren und Strings wie auch numerische Daten aus einem Datenstrom heraus zu extrahieren. [9], [81]

In dieser Anwendung wird einmalig der gesamte Sketch aufgezeigt und noch einmal die einzelnen Funktionen zum Verbindungsaufbau und die Verwendung des DHCP kurz erläutert. In den folgenden Anwendungsbeispielen wird eine aktive Verbindung vorausgesetzt und es werden nur noch spezifische Änderungen beschrieben.

Durch die ersten beiden Anweisungen werden die Header-Dateien für die SPI- und Ethernet-Bibliothek inkludiert.

```
#include <SPI.h>
#include <Ethernet.h>
```

Quellcode 13: Yahoo-Sketch - Header-Dateien für Bibliotheken inkludieren

Anschließend wird die MAC- und Server-Adresse definiert.

```
byte mac[] = { 0xDE,0xAD,0xBE,0xEF,0xFE,0xED };
char server[] = "search.yahoo.com";
```

Quellcode 14: Yahoo-Sketch - MAC- und Server-Adresse festlegen

Dann muss ein Objekt vom Typ »Ethernet-Client« erzeugt werden.

```
EthernetClient client;
```

Quellcode 15: Yahoo-Sketch - EthernetClient-Objekt erzeugen

Ebenfalls wird noch eine Variable erzeugt, welche das Ergebnis speichert.

```
int result;
```

Quellcode 16: Yahoo-Sketch - Variable zum Speichern des Ergebnisses

Anschließend wird die `setup()`-Funktion definiert und um das Ergebnis darstellen zu können, die Serielle-Kommunikation mit einer Baudrate von 9600 eingeleitet. Darauffolgend wird das Ethernet-Shield initialisiert und über DHCP automatisch konfiguriert. Durch die Bedingung wird im Falle einer fehlgeschlagenen Ethernet-Konfiguration über DHCP eine Fehlermeldung auf dem Seriellen-Monitor ausgegeben. Über die `delay()`-Funktion wird dem Ethernet-Shield Zeit zur Initialisierung eingeräumt.

```
void setup() {
    Serial.begin(9600);
   if(Ethernet.begin(mac) == 0) {
     Serial.println("Konfig fehlgeschlagen");
   }
   //1 Sekunde warten
   delay(1000);
}
```

Quellcode 17: Yahoo-Sketch - Initialisierung des Ethernet-Shields

Nach der Initialisierung wird in die `loop()`-Schleife gesprungen. Der restliche nun aufgeführte Code steht komplett in dieser Funktion.

```
void loop() {
```

Quellcode 18: Yahoo-Sketch - loop-Funktion

Durch eine weitere Bedingung wird eine Verbindung zum Yahoo-Server über Port 80 (HTTP) aufgebaut und überprüft ob der DNS-Server den Hostnamen in eine IP-Adresse auflösen konnte. Anschließend wird über die GET-Methode der HTTP-Request geschrieben.

```
if(client.connect(server,80)>0) {
    Serial.println("Verbindung erfolgreich");
    client.println("GET /search?p=50+km+in+mi HTTP/1.1");
    client.print();
}
else {
    Serial.print("Verbindung fehlgeschlagen");
}
```

Quellcode 19: Yahoo-Sketch - Verbindungsaufbau zum Yahoo-Server und HTTP-Request

Über weitere Bedingungen wird überprüft, ob der Client noch mit dem Server verbunden ist, anschließend wird über eine `find()`-Funktion nach einer Zeichenfolge gesucht. Mit der `parseInt()`-Funktion werden ab dem Gleichzeichen, die darauffolgenden Ziffern herausgefiltert. Dies geschieht so lange bis ein Zeichen kommt, welches keine Ziffer ist. Anschließend wird auf dem Seriellen-Monitor das Ergebnis ausgegeben. Ist der Client nach Beendigung des Suchprozesses mit dem Response durch und somit nicht mehr verbunden, wird der Client gestoppt beziehungsweise die Verbindung beendet und in eine `while()`-Endlosschleife gewechselt, damit die `loop()`-Funktion kein weiteres Mal ausgeführt wird. [9], [81]

```
  if(client.connected()) {
     if(client.find("Kilometers =")) {
       result = client.parseInt();
       Serial.print("50 km sind ");
       Serial.print(result);
       Serial.println(" Meilen.");
     }
     else {
       Serial.println("Kein Ergebnis.");
     }
   }
   else {
     Serial.println("Verbindung beendet.");
     client.stop();
     while(true);
   }
} //Ende der loop()-Funktion
```

Quellcode 20: Yahoo-Sketch - Parsen des HTTP-Response

Die serielle Ausgabe bei einem Request mit 50 Kilometern sieht hierbei wie folgt aus. Die `parseInt()`-Funktion konnte die nachfolgenden Ziffern herausfiltern und somit die Ausgabe wie gewünscht erstellen.

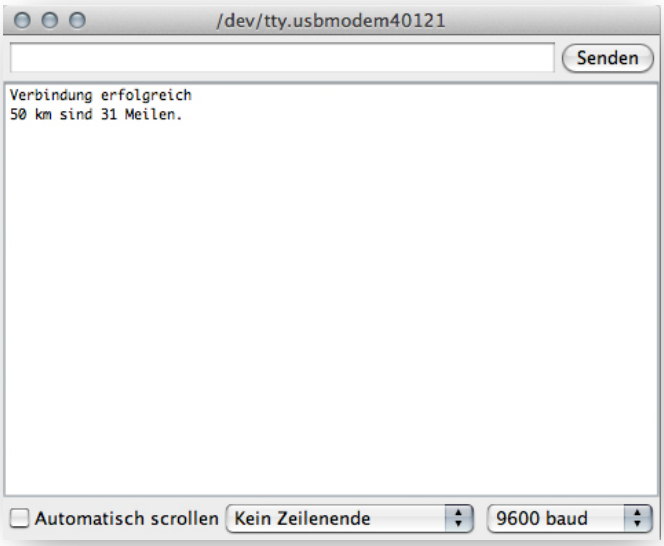


Abbildung 34: Yahoo-Sketch - Serieller Monitor Yahoo-Anfrage

Das Ergebnis wird hierbei mit Integer-Werten (ganze Zahlen) ausgegeben. Bei genaueren Werten muss eine andere Stream-Parsing-Funktion gewählt werden. Um Float-Werte (Gleitkommazahlen) zu generieren muss auf die `parseFloat()`-Funktion zurückgegriffen werden.

Im Anhang F »WiFi Web-Client Yahoo-Sketch« ist der Quellcode dieser Anwendung für das WiFi-Shield konfiguriert einsehbar.

Für gewöhnlich ändern Seiten wie Yahoo-Weather oder Yahoo-Finance, die für die Suche beziehungsweise Identifizierung relevanten HTML-Tags oder Zeichenfolgen nicht. Sollte dies jedoch vorkommen, so ist der oben dargestellte Code unbrauchbar und liefert keine Ergebnisse mehr. Daraufhin muss der Sketch bezüglich Änderungen durch die Seitenaktualisierung angepasst beziehungsweise korrigiert werden.

Eine wesentlich effizientere und konstantere Variante, um Daten von einem Web-Server abzurufen, bietet die Verwendung von Datenaustauschformaten wie XML und JSON. Im nächsten Kapitel wird dahingehend die Möglichkeit beschrieben, Daten mittels XML-API abzurufen. In dem Projekt zu Beginn des Recherchekapitels namens »Twitter-Mood-Light« wurde hingegen mit der JSON-Twitter-API gearbeitet, um die Tweets vom Server anzufragen.

2.8.5 XML/JSON-Daten abrufen

XML wird als Auszeichnungs- beziehungsweise Beschreibungssprache bezeichnet und dient der strukturellen Darstellung von Textdaten. Derartig formatierte Daten eignen sich zum Datenaustausch zwischen verschiedenen Systemen. Dabei werden diese von einem System bereitgestellt und können durch externe Anwendungen aufgerufen, gelesen und weiterverarbeitet werden. Zum Lesen werden hierzu Programme verwendet, sogenannte Parser, welche in vielen Programmiersprachen zur Verfügung stehen und eine schnelle Verarbeitung erlauben. Das XML-Format wird unter anderem bei RSS-Feeds eingesetzt und wird zum Beispiel auf Webblogs oder News-Seiten verwendet. [8]

Eine durchaus gängige Methode (wird auch bei der Apple Weather App verwendet), um aktuelle wie auch künftige Wetterdaten abzufragen, ist die Verwendung der Yahoo Weather RSS Feeds. Via GET-Methode wird ein HTTP-Request mit spezifischen Parametern für Lokation der erhobenen Daten und die Einheit (Fahrenheit oder Celsius) an die Yahoo-API, `http://weather.yahooapis.com/forecastrss`, gesandt. Zwei Parameter stehen anschließend zur Verfügung. [82]

- w: Parameter für den Ort, hier muss die ID (WOEID[55]) angehängt werden
- u: Parameter für die Einheit (Fahrenheit=f oder Celsius=c)

Berlin hat zum Beispiel den Identifier »638242«. Um nun das aktuelle wie auch das Wetter der nächsten Tage in Celsius abzufragen, muss die Anfrage wie folgt aussehen.

```
http://weather.yahooapis.com/forecastrss?w=638242&u=c
```

Der Response umfasst eine XML-Datei, welche über die Stream-Parsing-Funktion nach den für den Anwender interessanten Informationen, wie minimale und maximale Temperatur, durchsucht werden kann. Eine weitere sehr verbreitete Variante ist die Verwendung der TextFinder-Bibliothek und den dazugehörigen Funktionen, um mit einfachen Mitteln die gewünschten Werte aus der XML-Struktur zu extrahieren. Dementsprechende Arduino-Anwendungen arbeiten diesbezüglich nach der Client/Server-Kommunikation wie in Kapitel 2.8.4 beschrieben. [81]

Die Strukturen und Bezeichner unterscheiden sich je nach Entwickler und Dienst der Webseite, daher muss im Voraus nach einer gewünschten API gesucht und anschließend auf Anwendbarkeit für das eigene Projekt überprüft werden.

JSON-Daten abrufen

Das JSON-Format ist einfacher gestaltet und somit auch leichter zu schreiben und auszulesen. Des Weiteren reduziert JSON auch den Overhead, welchen XML mit sich bringt. Somit ist die angefragte Datenmenge eine geringere. Allerdings ist XML (Auszeichnungssprache) vielseitiger einsetzbar und schneller zu parsen als das Datenaustauschformat JSON. Daher bringen beide Vor- und Nachteile mit sich. Die Wahl des Formates wird oftmals durch die Betreiber einer Webseite eingeschränkt, so wird zum Beispiel bei Twitter für die Such-API seit Mai 2013 nur noch JSON unterstützt. [83]

Seit 2010 ist eine JSON-Bibliothek namens »aJson« für Arduino vorhanden. Diese ist noch nicht in die Arduino-IDE implementiert, da diese noch mit Fehlern behaftet ist. Allerdings können mit ihr relativ simpel JSON-Objekte erstellt, Zeichenfolgen untersucht und analysiert werden wie auch beim Parsen direkt gefiltert werden. Die Bibliothek und weitere Informationen zur Anwendung stehen auf folgender Webseite zur Verfügung, *https://github.com/interactive-matter/aJson*.

[55] Akronym für »Where on Earth Identifier«, Yahoo-spezifische IDs

2.8.6 Arduino als Web-Server

Ein Arduino kann nicht nur als Client fungieren, sondern auch als Server eingehende Anfragen bearbeiten und Informationen zurückliefern. In dieser Funktion muss der Arduino Webseiten ausliefern, um zum Beispiel Nutzern die Möglichkeit zu geben über einen Web-Browser Werte an den Pins einzusehen oder durch HTML-Formulare die Ein- und Ausgänge des Boards zu steuern. Arduino als Web-Server beinhaltet immer zwei Komponenten. Zum einen muss der Arduino als Server initialisiert werden, sodass über eine IP-Adresse auf eben diesen zugegriffen werden kann. Des Weiteren wird ein Client-Objekt erzeugt, welches den Arduino als Produzenten realisiert und gleichzeitig den Server darstellt. Der Produzent schreibt die HTML-Struktur, Inhalte und liest Werte an den Sensoren aus beziehungsweise schreibt Werte an die Pins, um Aktuatoren zu steuern. Insofern entsteht hierbei folgende Kommunikationsarchitektur mit einem Arduino-Board, welches von außerhalb als Server betrachtet wird, allerdings auch einen Client als Produzenten implementiert.

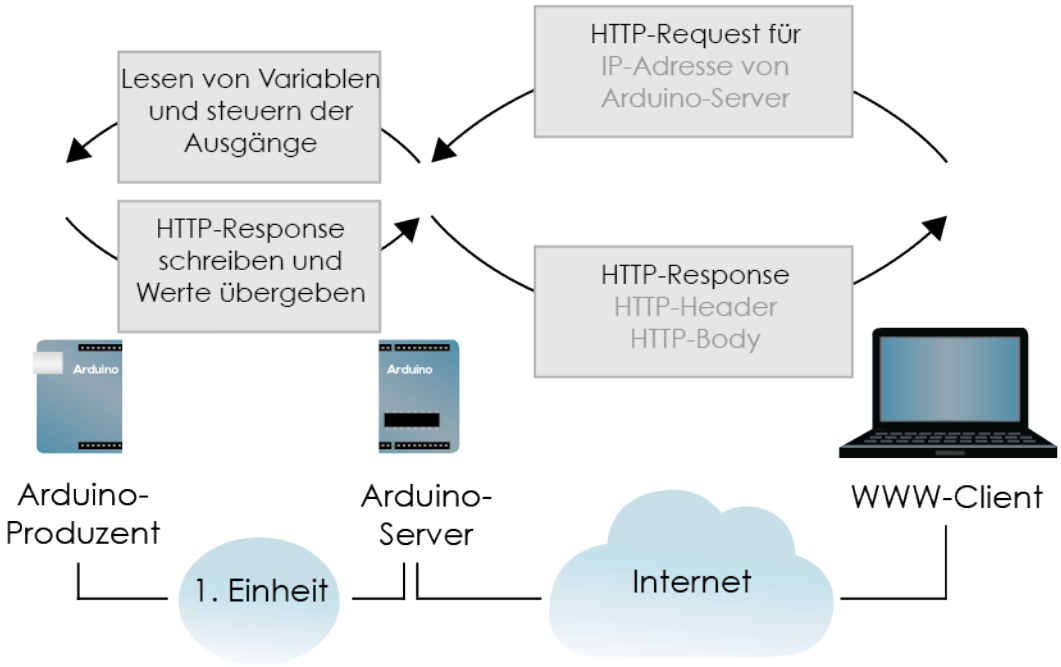


Abbildung 35: Arduino Web-Server Kommunikationsarchitektur

In diesem Kapitel wird vorausgesetzt, dass Grundkenntnisse über das Erstellen von einfachen HTML-Formularen vorhanden sind. In den folgenden beiden Erläuterungen werden Werte von den analogen Pins via Webseite ausgeben und Ausgänge über eine Webseite gesteuert. Auf den Arduino-Server kann nur aus dem lokalen Netzwerk zugegriffen werden. Über Port-Forwarding ist es auch möglich den Arduino-Server von überall beziehungsweise aus dem Internet (weltweites Netzwerk) aus anzusprechen. Dieses Szenario wird am Ende dieses Kapitels erklärt.

Werte der analogen Eingangspins per Webseite anzeigen

Wie zuvor als Web-Client muss auch hier zu Beginn des Sketches die SPI- und Ethernet-Bibliothek inkludiert werden. Eine IP-Adresse für den Arduino-Server kann entweder manuell eingetragen oder per DHCP zugewiesen werden. Diese ist nötig, um sich über den

Web-Browser mit dem Arduino-Server zu verbinden. Wird über DHCP gearbeitet, so muss über `Serial.println()` die zugewiesene IP-Adresse über den Seriellen-Monitor ausgegeben werden.

Im Setup des Sketches muss ein Objekt vom Typ »Ethernet-Server« erstellt werden mit der Angabe des zu verwendenden Ports. Bei HTTP-Anfragen wird standardgemäß Port 80 verwendet. Anschließend muss die Ethernet-Bibliothek initialisiert und der Server mit der IP-Adresse konfiguriert werden. Wird DHCP verwendet, so muss wie in Kapitel 2.8.3 »Ethernet-Shield mit automatischer IP-Adresse« vorgegangen werden.

```
//Server-Objekt erstellen
EthernetServer server(80);
//Ethernet initialisieren
Ethernet.begin(mac,ip);
//Server-Objekt initialisieren
server.begin();
```

Quellcode 21: Arduino-Server initialisieren

Die `loop()`-Funktion wartet auf Anfragen zum Web-Server. Ist ein Client verfügbar und es sind Bytes zum Lesen vorhanden, so liefert das Client-Objekt `true` zurück.

```
EthernetClient client = server.Verfügbar unter();
```

Quellcode 22: Arduino-Server - Client-Objekt erzeugen

Das Client-Objekt ist demnach ebenfalls der Web-Server, da es die eintreffenden Nachrichten für die IP-Adresse des Web-Servers verarbeitet. Über eine Bedingung kann abgefragt werden, ob ein Client erfolgreich gestartet wurde oder momentan kein Request an den Server eingeht und somit keine weiteren Operationen auszuführen sind bis auf das Wiederaufrufen der `loop()`-Funktion. Anschließend wird in eine `while()`-Schleife unter der Bedingung, dass der Client mit dem Server verbunden ist und Daten anfordert, gewechselt. Ist dies der Fall, werden über eine `read()`-Funktion die eintreffenden Bytes eingelesen bis das Zeilenende erreicht ist. Das Ende eines HTTP-Request kennzeichnet sich durch eine Leerzeile (Newline-Zeichen »\n«). Durch eine Bedingung zum Überprüfen des Endes des HTTP-Request kann anschließend der HTTP-Response-Header mit darauffolgender Leerzeile gesendet werden.

```
HTTP/1.1 200 OK
Content-Type: text/html
```

Quellcode 23: Arduino-Server - HTTP-Response-Header

Über `client.println()` wird der HTTP-Response-Header ausgegeben, gefolgt von den Werten der gewünschten Pins. Somit werden die gesamten Daten für den Response an den Server geschrieben, mit dem ein Client verbunden ist. Der Unterschied zwischen `client.print()` und `server.print()` liegt darin, dass unter der Client-Klasse nur Daten an den jeweilig anfragenden, verbundenen Client gesandt werden, während unter der Server-Klasse alle Gegenseiten der Verbindung den Response erhalten.

Ein Code-Beispiel zur Ausgabe von Werten der analogen Pins über einen Arduino als Web-Server ist in Anhang G »Ethernet Web-Server AnalogPin-Sketch« einzusehen. [5], [34]

Digitale und analoge Pins über eine Website steuern

Ein Arduino-Server kann nicht nur Webseiten übergeben, sondern auch auf GET- und POST-Statements reagieren und Aktionen ausführen. Hierzu muss, wie in der Erläuterung zuvor, eine Webseite an den Client übergeben werden. Diese Seite enthält ein Formular aus HTML-Tags, welche wiederum die Steuerelemente wie Buttons, Checkboxen, Labels usw. beinhaltet. Diese Elemente bilden die Schnittstelle für die Interaktion zwischen Benutzer und Server. Der Aufbau derartiger Programme ist vergleichbar mit vorheriger Erläuterung über Arduino als Web-Server. Allerdings werden hierbei über `client.print()` nicht nur Werte der Pins übergeben, sondern zusätzlich oder stattdessen ein oder mehrere HTML-Formulare.

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println();
client.println("<html>");
client.println("<form action'' method='get'>");
client.println("<input type='hidden' name='Pin13' value='1'/>");
client.println("<input type='submit' value='An'/>");
client.println("</form>");
client.println("<form action'' method='get'>");
client.println("<input type='hidden' name='Pin13' value='0'/>");
client.println("<input type='submit' value='Aus'/>");
client.println("</form>");
client.println("</html>");
```

Quellcode 24: Arduino-Server - HTML-Formular

Verwenden diese die GET-Methode, werden die Parameter bei einem Request an die URL angehängt.

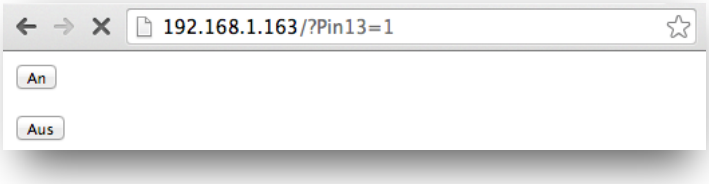


Abbildung 36: Arduino-Server - Browser-Screenshot

Über `find("GET /")` kann der Anfang des Request-Headers ermittelt und über Funktionen der Arduino-Stream-Klasse die Parameter gefunden werden. Anschließend wird über eine Stream-Parsing-Funktion der zugehörige Wert ausgelesen und man kann zum Beispiel einen digitalen Ausgang auf »HIGH« beziehungsweise »1« setzen.

```
if( client.find("GET /") ) {
  while(client.findUntil("Pin", "\n\r")){
    int pin = client.parseInt();
    int val = client.parseInt();
    pinMode(pin, OUTPUT);
    digitalWrite(pin, val);
  }
}
```

Quellcode 25: Arduino-Server - Parsen eines HTTP-Requests

Anders als bei der GET-Methode wird bei der POST-Methode der Query-String nicht in der URL mit übergeben, sondern im HTTP-Body (Nachrichtenkörper). Dadurch ist es möglich, wesentlich größere Datenmengen zu übermitteln. Der Arduino-Server muss hierbei nur nach dem Ende des HTTP-Headers suchen und anschließend nach den Parametern. Der Query-String ist ebenfalls einzeilig und somit kann über eine Bedingung das Ende eines solchen Strings gefunden und die Suche beendet werden. [5], [34]

```
client.find("\n\r"); //Ende HTTP-Header
if(client.findUntil("Pin", "\n\r")) {}
```

Quellcode 26: Arduino-Server - HTTP-Post

Bei beiden Methoden unterscheidet sich jeweils nur die Stelle im HTTP-Request, bei der nach den Parametern gesucht werden muss.

Port-Forwarding

Soll der Arduino-Server im Internet verfügbar sein, muss der Router, welcher als Schnittstelle zwischen lokalem Netzwerk und dem Internet fungiert, so konfiguriert werden, dass er eingehende Anfragen an den Arduino weiterleitet. Diese Technik wird »Port-Forwarding« beziehungsweise Portweiterleitung genannt und muss je nach Router spezifisch konfiguriert werden. [34]

Der Router wartet an einem bestimmten Port auf eintreffende Datenpakete aus dem Internet. Treffen dort Pakete ein, werden diese an einen bestimmten Computer beziehungsweise an einen anderen Port im internen Netzwerk weitergeleitet. Der Server kann nicht direkt aus dem Internet angesprochen werden, da aufgrund der begrenzten Anzahl an öffentlichen IP-Adressen nur der Router über eine derartige verfügt und innerhalb des lokalen Netzwerkes private Adressen vergeben werden, die nur innerhalb des Netzwerkes gültig sind. Somit steht bei Kommunikationen zwischen lokalen Netzwerken und dem Internet immer die IP-Adresse des Routers als Ziel- beziehungsweise Quelladresse im Header der Pakete. [80]

Das Verfahren, welches die Verwaltung und Weiterleitung von Paketen an den jeweils richtigen Port innerhalb eines Netzwerkes übernimmt, heißt NAT[56]. Heutzutage ist so gut wie jeder DSL-Router NAT-fähig und somit im Stande Netzadressen zu übersetzen. NAT macht nichts anderes als zum Beispiel die Sender-IP-Adresse von Paketen aus dem lokalen

[56] Akronym für »Network Address Translation«, wird benötigt um automatisch Adressinformationen in Datenpaketen so zu ersetzten, dass eine Kommunikation zwischen verschiedenen Netzwerken möglich ist.

Netzwerk durch die des Routers zu ersetzten, welche öffentlich ist und über das Internet geroutet werden kann. Um darauf zurückkehrende Pakete der richtigen Schnittstelle zuweisen zu können, werden Ports verwendet. Hierfür muss der Router eine NAT-Tabelle führen, in der er die IP-Adressen der Sender, Empfänger und den zugewiesenen Ports verwaltet. Somit wird zum Beispiel ein Server in einem lokalen Netzwerk über einen festgelegten Port aus externen Netzwerken eindeutig ansprechbar. Soll ein HTTP/TCP Port 80 Dienst auf einem Server über das Internet nutzbar sein, so muss der Client aus dem externen Netzwerk die IP-Adresse des Routers aufrufen. Der Router leitet dann die Anfrage für diesen Dienst (Port 80) an den entsprechenden Server im lokalen Netzwerk weiter. [34], [80]

Zur Konfiguration eines Routers sind drei Schritte notwendig.

1. **DHCP-Adresse reservieren**
   Um einem Gerät eine IP-Adresse zu reservieren, muss der Anwender auf der Router-Admin-Seite eingeloggt sein. In diesem Bereich findet sich eine Option zum Reservieren von IP-Adressen, welche der MAC-Adresse des Gerätes zugeordnet wird. Somit kann der Router jederzeit das spezifizierte Gerät erreichen und andere Clients, welche diesen Dienst nutzen, werden diese IP-Adresse nicht angeboten bekommen. Zum Testen einer erfolgreichen Konfiguration kann die IP-Adresse des Arduino-Web-Server auf dem Seriellen-Monitor ausgegeben werden. Bleibt die Adresse nach mehrmaligem Neustart des Programms dieselbe wie beim ersten Mal, so ist die IP-Adresse korrekt reserviert.

2. **Portweiterleitung**
   Nachdem eine feste IP-Adresse im lokalen Netzwerk für den Server vergeben ist, kann der eintreffende Web-Traffic zu dieser IP-Adresse gezielt weitergeleitet werden. Port 80 ist der Standard-Port für HTTP-Kommunikation und wird hierbei verwendet. Im Router-Administrationsbereich ist eine Option für Portweiterleitung auswählbar. In dieser muss der externe Port 80 des Routers, dem internen Port 80 für die reservierte IP-Adresse zugeordnet werden. Jede Anfrage aus dem externen Netzwerk, die an Port 80 des Routers ankommt, wird anschließend gezielt zum Arduino-Web-Server weitergeleitet.

3. **DNS Updating-Service**
   In den seltensten Fällen ist der Router über eine statische, globale IP-Adresse zu erreichen. Meistens werden dynamische IP-Adressen vergeben, die alle paar Tage oder Wochen vom jeweiligen Provider geändert werden. Daher müsste immer wieder überprüft werden, unter welcher Adresse der Router aus dem Internet zu erreichen ist. Für diesen Fall stehen dem Entwickler sogenannte dynamische IP-Services zur Verfügung. Dieser Service startet ein kleines Programm auf dem Router, welches periodisch die globale IP-Adresse prüft und zu einem entfernten Web-Server übermittelt. Der Router erstellt immer wieder eine Referenz von einer Domain auf die jeweilige dynamische, globale IP-Adresse. Ein etablierter Service in diesem Bereich heißt »DynDNS« und ist in vielen Routern schon vorkonfiguriert. Einige dynamische DNS-Services stehen zur freien Verfügung andere wiederum kosten geringe monatliche Beiträge. Nach diesem Schritt kann der Arduino-Web-Server aus dem Internet über eine Domain aufgerufen werden, welche sich nicht ändern wird und somit eine konstante Nutzung gewährleistet. [34], [84]–[87]

Die Konfiguration des Routers nimmt für unerfahrene Entwickler einige Zeit in Anspruch und ist in den meisten Fällen mit Kosten verbunden. Eine Lösung, bei der keine Netzwerk- und Router-Konfiguration notwendig ist, bietet »Teleduino«. Zur Verwendung von Teleduino muss auf der Webseite (*http://teleduino.org*) die gleichnamige Bibliothek geladen und in die Arduino-IDE implementiert werden. Um den Dienst zu verwenden, muss auf dem Arduino-Board mit Ethernet-Shield der mitgelieferte Proxy-Sketch geladen werden, welcher die Kommunikation mit dem Web-Server managt. Details zur Verwendung lassen sich auf der oben genannten Webseite von Teleduino nachlesen. [88]

2.8.7 E-Mail-Versand mit Arduino

Arduino verfügt zwar über keine E-Mail-Funktionalität, kann aber Daten und Informationen, welche für einen E-Mail-Versand relevant sind, zusammentragen und weiterleiten. Das Versenden von Mails kann über mehrere Möglichkeiten realisiert werden, folgend wird auf den direkten Versand von E-Mail-Informationen an einen Mail-Server und der Verarbeitung und Versendung über ein Webskript eingegangen. Voraussetzung für folgende Anwendungsszenarien ist eine aktive Internetverbindung aufseiten des Arduinos. Zum Austausch von E-Mails wird das SMTP[57]-Protokoll der Internetprotokollfamilie genutzt. Eine Verbindung zu einem SMTP-Server wird über Port 25 aufgebaut.

Direkter Versand

Beim direkten Versenden einer E-Mail über das Internet ist der Ablauf identisch zu dem eines E-Mail-Programms wie Apple-Mail. Durch eine Abfolge von Befehlen wird der Mail-Server definiert, kontaktiert und die einzelnen Befehle für den Versand gesendet.

Als Server muss die SMTP-Server-Adresse eingetragen werden. Wird zum Beispiel »O2 Online« als Provider verwendet, so lautet die SMTP-Adresse für den Postausgangsserver »*smtp.o2online.de*«. Als Port wird der Standard-SMTP-Port 25 gewählt.

```
//vor dem setup
char server[] = "smtp.o2online.de";

// im setup
if (client.connect(server, 25)) {}
```

Quellcode 27: Arduino-Mail - Verbindung zum Mail-Server

In der Bedingung kann anschließend der Mail-Header wie auch Textkörper geschrieben werden. Der Aufbau einer SMTP-Sitzung wird in Anhang H »Mail-Header und Textkörper« beschrieben.

[57] Akronym für »Simple Mail Transfer Protocol«

```
// Anfang Mail-Header
client.println("HELO Name");
client.println("MAIL FROM: absender@mail.de");
client.println("RCPT TO: empfaenger@mail.de");
client.println("DATA");
// Absender-Adresse
client.println("From: absender@mail.de");
// Empfänger-Adresse
client.println("TO: empfaenger@mail.de");
// Titel der Mail
client.println("SUBJECT: Arduino sendet E-Mail");
client.println();
// Anfang Mail-Textkörper
client.println("Das ist der Inhalt der Mail von Arduino.");
// Kennzeichnung Ende der E-Mail
client.println(".");
// Abmelden
client.println("QUIT");
```

Quellcode 28: Arduino-Mail - Mail-Header und Textkörper

Gemäß den Inhalten aus dem obigen Quellcodebeispiel kann die eintreffende Mail wie folgt aussehen.

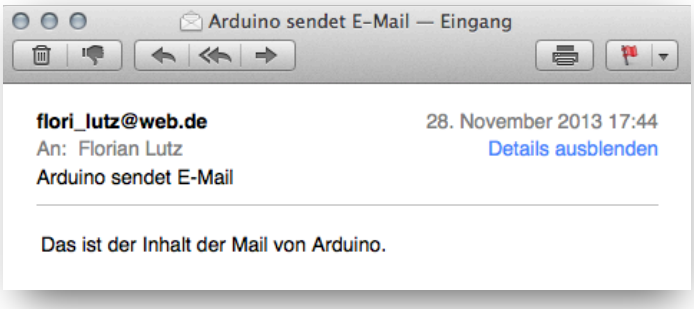


Abbildung 37: Arduino-Mail - Screenshot Apple-Mail

Die meisten Mail-Server benötigen für den E-Mail-Versand den Benutzernamen wie auch das zugehörige Passwort. In diesem Fall muss nach der »HELO« Anweisung eine zusätzliche Befehlszeile ergänzt werden.

```
client.println("HELO Name");
client.println("AUTH PLAIN XXXXXXXXX");
```

Quellcode 29: Arduino-Mail - Mail-Server Authentifizierung

Die Zeichenfolge »XXXXXXXXX« muss hierbei durch einen Base64-kodierten Benutzernamen/Passwort-String ersetzt werden. Arduino unterstützt allerdings kein SSL[58] und somit HTTPS, um das Passwort wie auch den Benutzernamen verschlüsselt zu übertragen. Daher ist dieser String so unsicher als würde man diese Daten per Klartext übermitteln. [8], [89]

[58] Akronym für »Secure Sockets Layer«, ist ein Verschlüsselungsprotokoll zu sicheren Datenübertragung

E-Mail-Versand über PHP-Skript

Eine E-Mail über ein PHP-Skript zu versenden ist eine wesentlich elegantere Variante, da PHP über Funktionen für den Mail-Versand verfügt. Hierzu wird allerdings ein Server benötigt, auf welchem das Skript abgelegt und vom Arduino-Client aufgerufen werden kann. Liegt dieses Skript auf dem Server, über den auch der Mail-Server läuft, sind in den meisten Fällen die Einstellungen für den Postausgangsserver bereits vorkonfiguriert und es kann direkt über eine Mail-Funktion gearbeitet werden. Ein entsprechendes PHP-Skript kann hierbei wie folgt aussehen.

```php
<?php
   // mailsender.php

   // Mailtext
    $mailtext = "Grußmail vom Arduino";

   // E-Mail versenden
   // mail(Empfänger, Betreff, Mailtext)
   $mail_sent = mail('meine@email.com', 'Arduino Mail', $mailtext);

   // Mailstatus ausgeben, 1: OK, 0: Fehler
   echo "Mail Status: ".$mail_sent;
?>
```

Quellcode 30: PHP-Skript Mail-Sender

Nach Aufruf des Skriptes wird automatisch eine Mail an den Empfänger (erster Parameter in der `mail()`-Anweisung) gesendet. Der Skriptaufruf findet über die URL statt und kann von einem Arduino über die GET-Methode als Parameter an die URL des Servers angehängt werden. Der Aufruf einer Webseite unter Verwendung der GET-Methode ist in Kapitel 2.8.4 »Arduino als Web-Client« beschrieben. [8]

2.8.8 Arduino nutzt Twitter

Die Kommunikationsplattform »Twitter« kann von Arduino für diverse Anwendungen verwendet werden. Dabei kann Arduino als Twitter-Client Kurznachrichten senden oder zum Beispiel auf Hashtags lauschen. Beide Möglichkeiten bilden eine Kommunikationsschnittstelle zwischen dem Internet und dem Arduino. Somit wird eine schon bestehende Plattform genutzt, um zum Beispiel Sensordaten wie Temperaturwerte zu übermitteln. Des Weiteren kann der Arduino auch auf Tweets reagieren und Ausgänge steuern.

Aufgrund der sich Mitte 2013 ändernden Konfigurationen für die Authentifizierung bei Twitter, wie auch durch Umstellungen der APIs, ist eine reine Arduino-basierende Programmierung schwierig und anfällig gegenüber Änderungen. Die in der Arduino-Entwicklungsumgebung vorhandenen Beispiel-Sketche sind veraltet und verwenden nicht das »OAuth«-Protokoll für die sichere API-Autorisierung, welche mittlerweile von Twitter verlangt wird. Sven Hoffmann, Entwickler im Bereich Interaction-Design, hat eine Bibliothek (nur für Ethernet-Shield) geschrieben, welche die benötigte Autorisierung implementiert. Diese kann auf »*http://hofmannsven.com/2013/laboratory/arduino-twitter-library/*« heruntergeladen und in das Bibliotheksverzeichnis von Arduino eingefügt werden.

Bei erfolgreicher Implementierung der Bibliothek sind nach Neustart von Arduino zwei Beispiel-Sketche vorhanden. Einer der beiden ist für das Versenden von Strings als Tweet aus dem Programmcode zuständig, der andere hingegen für das Übermitteln von Nachrichten, die über den seriellen Port eingelesen werden. Der Sketch benötigt zusätzlich den »OAuth-Token[59]«, welcher über »*http://arduino-tweet.appspot.com/oauth/twitter/login*« angefordert werden kann. Durch dieses Verfahren wird dem Arduino Zugriff auf den Twitter-Account gewährleistet. Die entsprechende Bibliothek für das Arduino-WiFi-Shield kann unter »*http://www.instructables.com/id/How-to-tweet-from-an-Arduino-using-the-wifi-sheild/step4/*« heruntergeladen werden. [90], [91]

Eine wesentlich komfortablere Lösung bietet hierzu »ThingSpeak«, eine Plattform zum Erstellen von »Internet of Things«-Applikationen. Bei einer Kommunikation über ThingSpeak als Proxy für Twitter ist keine Implementierung der OAuth-Autorisierung in den eigenen Programmcode notwendig. Es muss lediglich ein kostenloser Account auf der Seite angelegt werden, über den anschließend ein API-Key angefordert werden kann. Über diesen Key können vier verschiedene Applikationen, zwei spezifisch für Twitter, realisiert werden.

- **ThingTweet**
  Verbindet den Twitter-Account mit ThingSpeak, sodass über ein Arduino Nachrichten an Twitter übermittelt werden können.
- **TweetControl**
  Hört auf Befehle oder Anweisung von Twitter und führt daraufhin definierte Aktionen aus.

Ebenfalls ist eine Autorisierung des Twitter-Accounts notwendig, damit der ThingSpeak-Dienst Nachrichten an den Account senden darf. [5], [92]

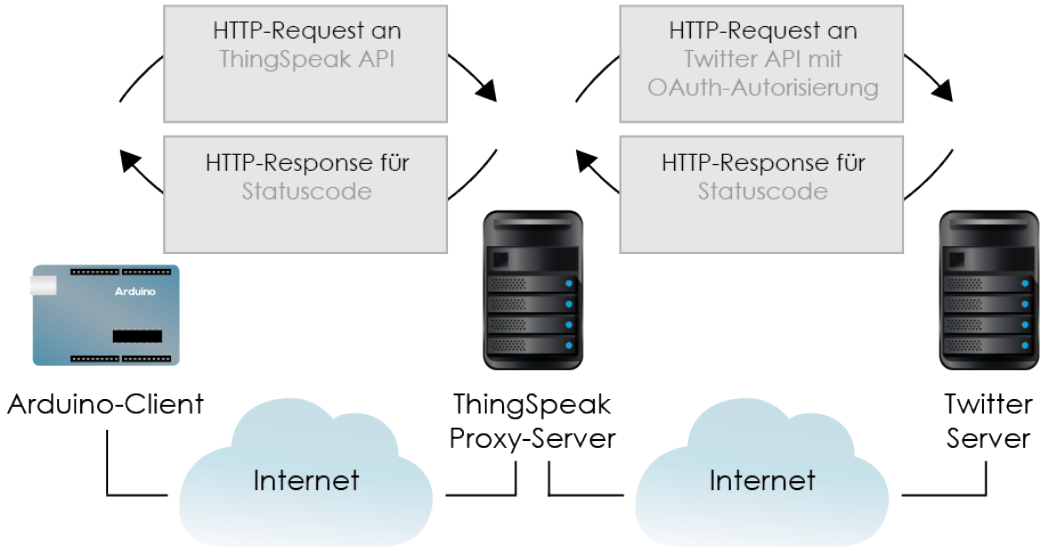


Abbildung 38: Kommunikationsarchitektur, Arduino twittert über ThingSpeak-Proxy

[59] Eine Zeichenkette welche an Stelle von Benutzername und Passwort verwendet wird.

2.8.9 UDP statt TCP

Bei der Verwendung beider Protokolle treten Unterschiede auf der Transport- wie auch auf der Anwendungsschicht des OSI-Modells[60] auf. TCP und UDP sind auf der Transportschicht angesiedelt, und während TCP Einheiten als Segmente übermittelt, verwendet UDP die Datagrammvermittlung. Diese ist wesentlich schneller, da ein Paketverlust ignoriert wird. TCP hingegen verfügt über verschiedene Möglichkeiten der Prüfung und Behebung von Paketverlusten, was allerdings die Bandbreite verringert. UDP ist diesbezüglich eine schnelle und einfache Möglichkeit, Nachrichten über das Ethernet oder WLAN zu senden und empfangen. Allerdings kann bei der Zustellung der Nachrichten nicht gewährleistet werden, dass diese überhaupt und in richtiger Reihenfolge den Empfänger erreichen. Für Statusmeldungen oder Ausgaben von Werten der Arduino-Sensoren eignet sich hingegen dieses Protokoll, da verlorene Nachrichten einfach durch Darauffolgende ersetzt werden. [80]

Auf der Anwendungsschicht sind Protokolle angesiedelt, welche das Format für Nachrichten und Informationen für jede bestimmte Anwendung definieren. In Abbildung 39 befinden sich die beiden Protokolle UDP und TCP auf der Transportschicht, darüber die meist bekannten Protokolle SMTP, FTP und HTTP, welche alle drei TCP verwenden. RTSP ist ein Streaming-Protokoll und verwendet dahingehend UDP. [3], [5], [93]

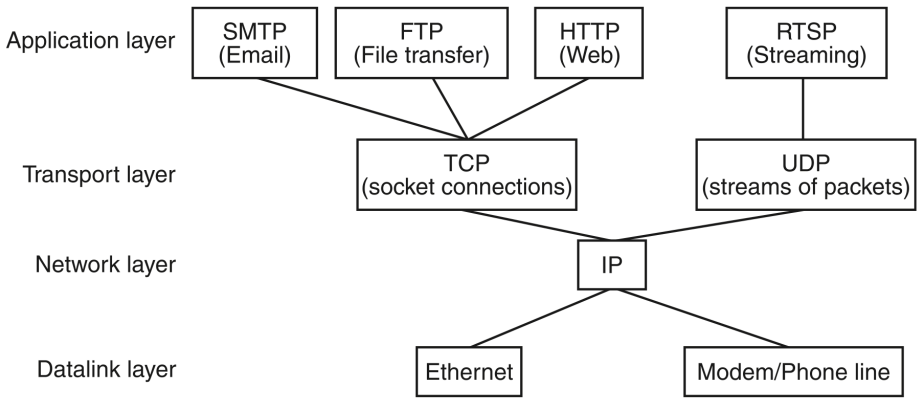


Abbildung 39: Die Protokolle zwischen dir und dem Web [3]

Zur einfachen Anwendung bietet Arduino hierzu WiFi- oder Ethernet- spezifische UDP-Bibliotheken mit eingebetteten Anwendungsbeispielen an.

2.8.10 Cloud-Service »Xively« zur Daten-Visualisierung

Xively ist ein kostenloser (nicht bei kommerzieller Nutzung) Cloud-Service[61], welcher speziell für »Internet of Things«-Anwendungen entwickelt wurde. Den Nutzern sollen hierdurch infrastrukturelle Barrieren bei der Entwicklung von vernetzten Produktlösungen genommen werden und somit den Fortschritt und innovative Lösungen fördern. Xively stellt Entwicklern eine Computer-Plattform in einer Cloud zur Verfügung, welche

[60] Referenzmodell für Netzwerkprotokolle als Schichtenarchitektur

[61] Ein Cloud-Service stellt abstrahierte IT-Infrastrukturen dynamisch an den jeweiligen Bedarf angepasst über ein Netzwerk zur Verfügung. [94]

webbasierte Tools und Ressourcen für Entwickler bereitstellt, die mit geringem administrativen Aufwand und ohne Anschaffungskosten durch Soft- und Hardware genutzt werden können. Bereitgestellte Bibliotheken und Skripte für verschiedenste Plattformen und Programmiersprachen, wie Arduino, Android, Java, PHP und viele mehr lassen die Xively-API reibungslos client- und oder serverseitig integrieren. Bei der Entwicklung eines Prototypen wird hingegen nur ein kleiner Bereich des umfangreichen Angebots dieses Services genutzt, da die Xively-API auch die Integration von Zahlungssystemen, Produktverwaltung und vieles mehr unterstützt. [95]

Xively Daten-Hierarchie

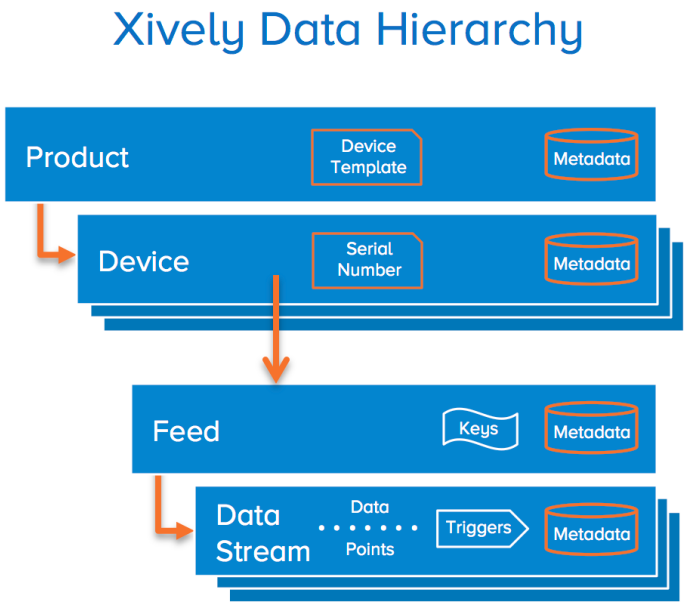


Abbildung 40: Xively Daten-Hierarchie [96]

Ein Produkt kann mehrere Geräte umfassen, welche jeweils über einen Feed verfügen. Dieser Feed umschließt ein oder mehrere Datenstreams. Bei der Entwicklung eines Prototypen, welcher sich rein mit der Visualisierung und der Datenspeicherung befasst, muss kein Produkt (Product-Ebene) entwickelt werden. Diese Ebene setzt sich mit der Verwaltung verschiedenster Systeme auseinander, die für eine Testintegration des eigenen Systems in einen Visualisierung-Service irrelevant sind.

Sensordaten von Arduino übermitteln

Sensordaten können auf verschiedene, wie in den Kapiteln zuvor erläuterten, Varianten einem Client übermittelt werden. Hierzu kann der Arduino als Web-Server selbst eine Webseite als HTTP-Response übergeben, was allerdings eine Datenspeicherung und Visualisierungen aufgrund der geringen Speicherkapazität schwierig gestaltet, wie auch das Ansprechen des Servers aus dem Internet. Eine weitere Möglichkeit ist die Übermittlung von Daten beziehungsweise Informationen durch einen Client an ein sich auf einem externen Server befindlichen Skript.

Xively ermöglicht Arduino als einfacher HTTP-Client zu fungieren und lediglich HTTP-Request mit einem oder mehreren Sensorwerten an den Service zu übertragen. Dieser nimmt dem Entwickler anschließend die aufwändige Arbeit der Datenspeicherung und Visualisierung ab. Des Weiteren ermöglicht Xively auch das Sammeln, Archivieren und Visualisieren von ganzen Sensornetzen. Die einzelne Einheit bildet hierbei einen Sensorknoten, welcher durch Sensoren Werte der Umgebung erfasst, mit einem Prozessor verarbeitet und anschließend über ein Netzwerk an eine zentrale Einheit (Xively) sendet. Aufgrund der hier zu verwendenden Kommunikationsarchitektur benötigen diese Systeme keine Portweiterleitung, Speicherung der IP-Adresse oder weitere Konfigurationen. Die in diesem Kapitel verwendete Online-Grafik-Schnittstelle namens Xively erleichtert die Entwicklung von Live-Datenvisualisierung und Datenlogger ausgehend von einem Arduino-System. Das Überwachen von Daten, auch Monitoring genannt, ist ein oft genutztes Anwendungsszenario, welches in Verbindung mit der Xively-API umgesetzt wird. Hierbei kann zum Abfragen der Daten und Informationen auch auf die von Xively bereitgestellte Smartphone-Applikation zurückgegriffen werden. [95]

Um diesen Dienst nutzen zu können und Live-Daten-Feeds an Xively zu übermitteln, wird ein kostenloser Account auf der gleichnamigen Webseite benötigt. Ist dieser erfolgreich erstellt und verifiziert, kann unter dem Konto des Nutzers bei »Development Devices« ein neues Gerät hinzugefügt werden. In diesem Prozess muss für den Feed ein Bezeichner, eine Beschreibung, und ob es sich um einen öffentlichen oder privaten Feed handelt, angegeben werden. Nach der Bestätigung wird das Gerät angelegt und man gelangt auf eine Projektseite, die alle wichtigen Informationen bereitstellt, die später in einem Arduino-Sketch relevant für die Kommunikation zwischen Board und Xively-Service sind. Auf der Webseite wird ebenfalls die entsprechende Bibliothek für Arduino bereitgestellt. Diese benötigt des Weiteren die HTTPClient-Bibliothek, welche unter dem Download-Verzeichnis der Xively-Bibliothek auf GitHub verlinkt ist. Die Bibliothek bindet zudem Beispiel-Sketche in die Arduino-Entwicklungsumgebung ein, sodass Datenupload-Stream auf den Server, Datendownload-Stream von Xively wie auch der Upload mehrerer Sensorwerte durch geringe Änderungen am Sketch durchgeführt werden können. Hierzu werden die Beispiel-Sketche jeweils für das Ethernet- wie auch WiFi-Shield direkt angeboten.

Um einen Feed auf dem erstellten Xively-Account ansprechen zu können, müssen daher lediglich drei Werte verändert werden. Die Konfiguration für die Kommunikation im eigenen Netzwerk, wie in Kapitel 2.8.3 »Ethernet-Shield einrichten« beschrieben, der Zugriff auf die Xively-API über den auf der Webseite generierten API-Key und die Feed-ID, welche ebenfalls auf der Webseite zu finden ist und automatisch generiert wurde. Wird der Sketch auf dem Arduino ausgeführt, so wird eine Verbindung zum Xively-Server aufgebaut und auf der Webseite werden nach und nach Sensorwerte grafisch angezeigt oder der aktuelle Wert dargestellt. Über einen Request-Log können die jeweiligen eingegangen Requests verfolgt werden.

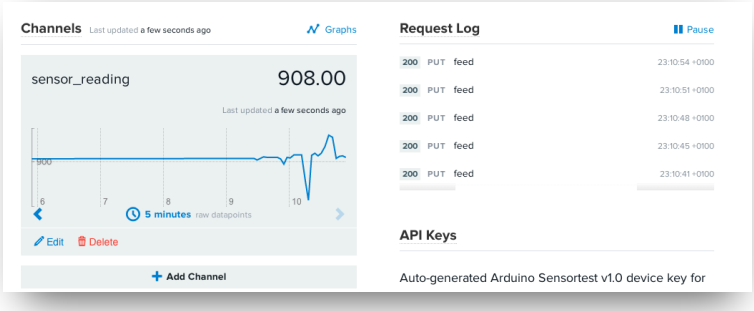


Abbildung 41: Arduino-Xively - Xively-Sensortest

Im Programmcode wird hierzu ein Objekt-Array namens `datastreams[]` erstellt. Dieser enthält Objekte, welche alle Informationen des Feeds beinhalten wie den Sensornamen und den zugehörigen Datentyp. Anschließend wird der Feed in ein `XivelyFeed`-Objekt eingehüllt, welches die Feed-ID, die Datenstream-Informationen und die Anzahl der sich im Array befindenden Objekte enthält. Die Daten werden in einem variablen Zeitintervall an die Xively-API im JSON-Format via HTTP-Request übermittelt (siehe Anhang I: Xively Request-/Response-Header).

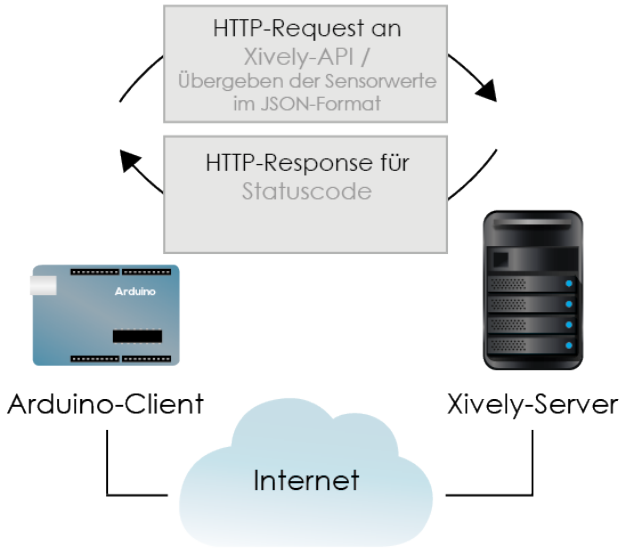


Abbildung 42: Kommunikationsarchitektur - Arduino-Client/Xively-Server

Die Xively-API unterstützt zwei weitere Formate, XML und CSV. Das Standard- beziehungsweise Default-Format ist hingegen JSON und wird immer dann verwendet, wenn kein anderes Format angegeben wird. Das JSON-Format eignet sich in diesem Anwendungsbeispiel aufgrund des geringen Overheads gegenüber XML und der damit verbundenen benötigten Bandbreite zum Übermitteln der Daten. Des Weiteren können Informationen dieses Formats einfach mit JavaScript in den Browser analysiert beziehungsweise geparst werden. [95]

Mehrere Sensoren verwenden

Wie erwähnt, werden die Daten über Datenstream-Objekte verwaltet, welche wiederum einem Array zugeordnet sind. Daher muss für einen weiteren Sensor ein eigenes

Datenstream-Objekt angelegt werden, welches über eine Feed-ID und den jeweiligen Datentyp verfügt. Der Datenstream-Array wird anschließend einem Feed-Objekt zugeordnet, hierbei muss ausschließlich die Anzahl der in dem Array vorkommenden Feed-Objekte, angegeben werden. Der Programmcode kann hierbei wie folgt aussehen und ist beliebig erweiterbar.

```
XivelyDatastream datastreams[] = {
  XivelyDatastream(lightId, strlen(lightId), DATASTREAM_FLOAT),
  XivelyDatastream(tempId, strlen(tempId), DATASTREAM_FLOAT),
};
XivelyFeed feed(2124383220, datastreams, 2);
```

Quellcode 31: Arduino Xively - Xively-Datenstreams

Im Entwicklungsbereich auf der Xively-Webseite kann unter dem Geräteverwaltungssystem die eingehenden Requests eingesehen und Visualisierungen angezeigt werden. Es müssen auf der Webseite unter dem jeweiligen Gerät keine weiteren Konfigurationen vorgenommen werden, wenn im Arduino-Sketch zusätzliche Datenstreams hinzufügt werden. Die Xively-API bekommt dies über den HTTP-Request vom Arduino-Client mitgeteilt und passt automatisch die Webseite an die neuen Streams an.

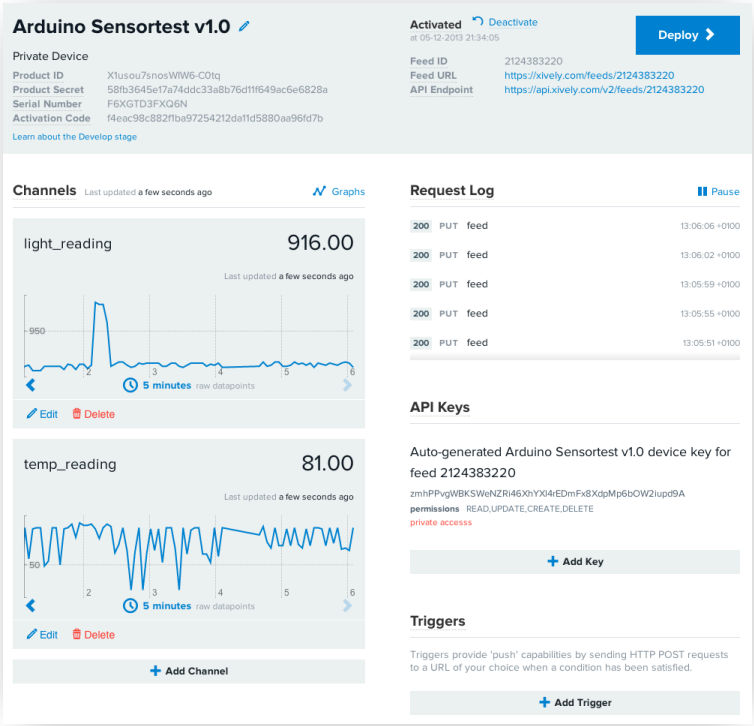


Abbildung 43: Arduino-Xively - Sensortest Temperatur und Licht

2.9 Lokales Datenlogging und Visualisierung

Nicht für jeden Anwendungsfall müssen Daten an eine Service-Plattform im Internet übermittelt werden und überall abrufbar sein. Oftmals genügt das Sammeln, sogenanntes Logging von Daten auf lokaler Ebene. Hierbei muss unterschieden werden, ob Daten auf einem Speichermedium direkt auf dem Arduino-Board geloggt oder direkt an einen Computer gesendet werden sollen.

2.9.1 Logging auf SD-Karten

Einige Arduino-Shields sind von vorneherein mit einem SD-Kartenslot ausgestattet und können über eine entsprechende Bibliothek genutzt werden. Das Arduino WiFi-Shield verfügt unter anderem über eine derartige Schnittstelle und kann somit, neben der Funktion als Schnittstelle zur Netzwerkkommunikation, auch Daten direkt lokal speichern. Des Weiteren existieren diverse Shields von Drittanbietern, welche zusätzlich mit einer Batterie-gepufferten Real-Time-Clock ausgestattet sind und dadurch Datum und Zeit ebenfalls mit abspeichern können. Ist dies nicht der Fall, kann über weitere Hardwareerweiterungen diese Funktionalität herbeigeführt werden.

Die SD-Bibliothek bietet hierzu zwei verschiedene Klassen an.

- **SD-Klasse**
  Diese Klasse ist für den Zugriff auf die SD-Karte zuständig und stellt dementsprechende Funktionen zur Verfügung. Des Weiteren können Dateien und Verzeichnisse bearbeitet und verwaltet werden.
- **File-Klasse**
  Durch die Funktionen dieser Klasse kann auf die Karte geschrieben oder Dateien von dieser gelesen werden. [97]

Die Informationen können zum Beispiel in einem CSV-Datenformat abgelegt werden, in dem die einzelnen gelesenen Werte kommasepariert gespeichert werden. In einer Schleife können so kontinuierlich Werte und Zeitangaben in die Datei geschrieben werden.

```
file.print(millis());
file.print(',');
file.print(analogRead(0));
file.print(',');
file.print(analogRead(1));
```

Quellcode 32: Datalogging analoger Pins

Durch Tabellenkalkulationsprogramme wie zum Beispiel Microsoft Excel können die Informationen anschließend wiederum ausgelesen und dargestellt werden. Hierzu ist es allerdings nötig, die SD-Karte vom Arduino Board zu entfernen und in den entsprechenden Computer zur Übertragung der Dateien zu stecken. Dadurch ist keine Echtzeit-Visualisierung der Daten möglich, außer wenn durch eine weitere Kommunikationsvariante die Daten zusätzlich an einen Server oder Endgerät gesendet werden.

2.9.2 Direkte Kommunikation mit dem Computer

Ist das Speichern auf einer SD-Karte nicht erforderlich, können Daten auch direkt an einen Computer übermittelt werden. Mittels einer objektorientierten Programmiersprache mit integrierter Entwicklungsumgebung namens »Processing« können Daten, welche von einem Arduino übermittelt werden, in einer Datei abgelegt und visualisiert werden. Der Arduino sendet hierbei Daten an einen seriellen Port an dem Computer und die Processing-Umgebung kann mittels einer Bibliothek für serielle Kommunikation wiederum den Port abhören und eintreffende Bytes lesen. Um eine Computer-zu-Arduino-Kommunikation aufzubauen, eignen sich die in der Arduino-Software-Distribution enthaltene Firmata-Bibliothek und die entsprechenden Beispiel-Sketche. Diese ermöglichen es, das Arduino-Board aus Processing heraus zu steuern ohne einen Programmcode für das Arduino zu schreiben beziehungsweise zu konfigurieren. Für Processing muss hierzu die Arduino-Bibliothek eingefügt werden und anschließend kann ebenfalls ein Beispiel-Code für die Kommunikation mit einem Arduino-Board aufgerufen werden. In diesem muss noch der serielle Port, an dem das Arduino angeschlossen ist, ausgewählt werden. Die Funktionen, welche in Processing dadurch zu Verfügung stehen, haben meist den gleichen Bezeichner wie in der Arduino-Umgebung. Im folgenden Beispiel wird eine Leuchtdiode, angeschlossen an Pin D13 des Arduino-Boards, in einer Schleife für eine Sekunde auf »HIGH« und anschließend für die gleiche Zeitspanne auf »LOW« gesetzt. Hierbei sind die Parallelen in der Programmierung zwischen Arduino und Processing klar zu erkennen. [5]

```
import processing.serial.*;
import cc.arduino.*;

Arduino arduino;
int ledPin = 13;

void setup()
{
  arduino = new Arduino(this, Arduino.list()[0], 9600);
  arduino.pinMode(ledPin, Arduino.OUTPUT);
}

void draw()
{
  arduino.digitalWrite(ledPin, Arduino.HIGH);
  delay(1000);
  arduino.digitalWrite(ledPin, Arduino.LOW);
  delay(1000);
}
```

Quellcode 33: Processing-Sketch, serielle Kommunikation [98]

In diesem Beispiel-Code wird über `digitalWrite()` dem Pin des Arduino-Boards ein Wert zugewiesen. Über `digitalRead()` oder `analogRead()` kann hingegen ein Wert, welcher durch einen Sensor erfasst wird, eingelesen und anschließend dargestellt werden.

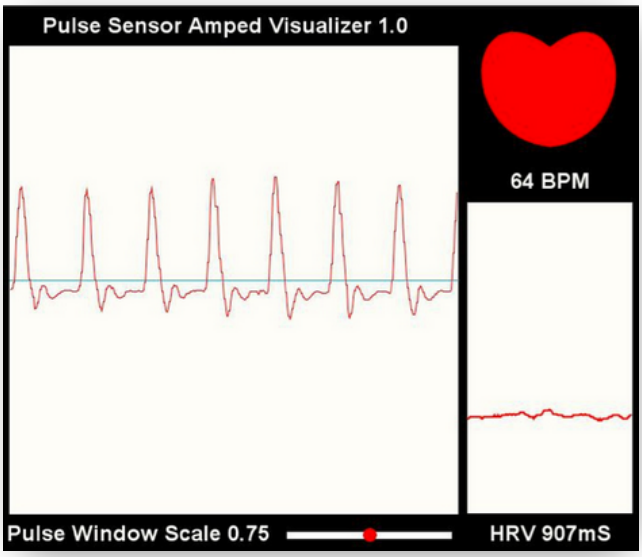


Abbildung 44: Processing-Visualisierung [99]

In dieser Abbildung wird über ein Arduino-Board und einen Pulssensor der Herzschlag über Processing am Computer visualisiert. In vielen Fällen ist es zudem nützlich, die gewonnen Informationen zusätzlich in einer Datei zu speichern und somit längerfristig bereitzustellen. Hierfür bietet Processing ebenfalls Funktionen an, welche zum Erzeugen und Beschreiben von Dateien verwendet werden können. Die Wahl des Dateiformats und die Formatierung der Informationen können hierbei frei gewählt und entsprechend der weiteren Verwendung angepasst und gespeichert werden.

2.10 Monitoring im Gesundheitswesen

Dieses Kapitel setzt sich mit der Erfassung von Sensordaten im Bereich Gesundheitswesen auseinander. Die Recherche bezieht sich hierbei auf das Krankheitsbild episodischer und paroxysmaler[62] Krankheiten des Nervensystems, speziell auf tonisch-klonische Anfälle, einer Anfallsform im Bereich der Epilepsie.

2.10.1 Tonisch-klonischer Anfall

Diese Form eines epileptischen Anfalls wird auch »Grand-mal-Anfall« genannt und bringt zweifellos die dramatischten Symptome im Bereich der Neurologie hervor. Eingeleitet wird dieser durch plötzlichen Bewusstseinsverlust begleitet von massiven Muskelspannungen der gesamten Muskulatur. Aufgrund der anhaltenden Verkrampfungen auch der Schlund- und Atemmuskulatur kommt es zu Sauerstoffunterversorgungen, worauf das Herz mit einer erhöhten Schlag-Frequenz reagieren muss, um den Körper weiterhin mit Sauerstoff zu versorgen. Anschließend geht der Anfall in Muskelzuckungen über, vor allem an Beinen und Armen, was mehrere Minuten andauern kann. [100], [101]

[62] gleichbedeutend wie »anfallsartig«

Zu lange anhaltende Anfälle können im schlimmsten Fall zu Atmungs- und Herzstillstand führen. Die meisten derartigen Anfälle treten hierbei nachts im Schlaf auf und zeigen ein eindeutiges Muster, bei dem nach einem Krampfanfall eine Phase mit schneller Atmung und abflachender Hirnaktivität folgt. Daraufhin kam es bei einem Drittel der in einer Studie untersuchten Patienten zu einem Atmungs- und Herzstillstand, welcher tödlich endete. Bei den übrigen Patienten setzte hingehen die Atmung und der Herzschlag wieder ein. Spätestens nach 11 Minuten nach dem Krampfanfall, traten auch bei diesen Patienten dauerhafter Atmungs- und Herzstillstand ein. [101]

> „Eine bessere Überwachung von Epilepsiepatienten ist unbedingt notwendig, um rechtzeitig lebenserhaltende Maßnahmen einleiten zu können."
> Professor Andreas Schulze-Bonhage [101]

Professor Schulze-Bonhage weist zusätzlich darauf hin, dass durch eine ständige Messung von Sauerstoffsättigung, Puls und Herzfrequenz derartige Anfälle frühzeitig erkannt und somit der tödliche Ausgang verhindert werden kann. [101]

Auswirkungen von nächtlicher Überwachung bei Kindern

Neben den Risiken für den Patienten an sich, wirkt sich die Diagnose Epilepsie auch auf den Gesundheitszustand der Eltern aus. Studien zeigen, dass Eltern, deren Kinder unter epileptischen Anfällen leiden, mehrmals in der Nacht den Zustand ihres Kindes überprüfen. Dies führt zu deutlich reduzierter Schlafqualität aufseiten der Eltern. Allerdings führt auch oftmals ein Fehlalarm von Monitoring-Geräten zur Überwachung des Kindes, zu zusätzlichem Stress und Schlafunterbrechungen. Dadurch entstehen wiederum höhere Depressions- und Ängstlichkeitswerte bei den Eltern. Zur nächtlichen Überwachung sind auf dem Markt schon ein paar wenige Systeme verfügbar. Hierzu im nächsten Kapitel mehr. [102]

2.10.2 Systeme zur Überwachung

Auf dem Markt sind verschiedene Geräte zum Überwachen von Körperfunktionen vorhanden und teilweise etabliert. Einige davon werden bei Patienten mit epileptischen Anfällen eingesetzt, welche allerdings nur Systeme zur Überwachung von Vitalfunktionen wie Sauerstoffgehalt und Puls sind. Bei der Recherche nach Monitoring-Geräten für Epileptiker für den Hausgebrauch wurde auf Foren und Blogs im Internet zurückgegriffen. In diesen tauschen sich Eltern von betroffenen Kindern über verschiedene Geräte aus und empfehlen oder warnen vor gewissen Eigenschaften und Handhabungen. Relevant bei diesen Systemen ist eine Warnfunktion, um bei auffälligen Werten bei der Messung, Alarm schlagen zu können. Des Weiteren ist es wichtig, die gewonnenen Daten speichern zu können, um somit dem behandelnden Arzt die Möglichkeit zur Auswertung relativ genauer Werte zu ermöglichen.

VitaGuard VG310

Dieses System ist ein Vitalfunktions-Monitor und zur Überwachung von Sauerstoffsättigung und Puls einsetzbar. Es verfügt über eine Akku-Laufzeit von acht Stunden und ist für den Einsatz im häuslichen Bereich vorgesehen. Bei Über- oder Unterschreiten eingestellter Grenzwerte kann das System akustischen und optischen Alarm auslösen. Zusätzlich kann dieses Gerät über einen Anschluss mit weiteren externen

Alarmgebern, wie zum Beispiel um eine Schwesternrufanlage erweitert werden. VitaGuard verfügt des Weiteren über einen internen Speicher, der mit einem Mini-USB ausgelesen werden kann. Auf diesem Speicher werden Alarm-Ereignisse aufgezeichnet und können somit von einem Arzt ausgewertet werden. Das System wiegt 700 Gramm und wird mit einer Auswertsoftware ausgeliefert, sodass die sonst auf dem Display angezeigten Signalkurven auch längerfristig gespeichert und analysiert werden können. Der VitaGuard VG310 wird Patienten von Ärzten verschrieben. [103]

Epi-Care 3000

Ein weiteres System ist der Epi-Care 3000, welcher laut Hersteller, die Gefahr unentdeckter klonischer Anfälle nahezu ausschließt. Auch dieser verfügt über eine Alarmerweiterung und kann akustische und optische Signale aussenden. Während der VitaGuard auch für den täglichen Gebrauch entwickelt wurde, ist der Epi-Care hingegen ausschließlich nachts einsetzbar, da dieser über einen Drucksensor verfügt, der unter der Matratze angebracht wird und Muskelverkrampfungen misst. Dieser ist so programmiert, dass er einen epileptischen Anfall von normalen Schlafbewegungen unterscheiden kann. Allerdings muss er je nach Matratzen-Typ und Person eingestellt und installiert werden. Über einen internen Speicher werden die auftretenden Anfälle mit Zeitpunkt, Dauer und Stärke dokumentiert. Dieses System bedarf einer Einweisung der Betreuungsperson in den Umgang mit den Alarmfunktionen. Ein Vorteil ist hierbei, dass der Patient nicht durch ein Gerät beeinflusst wird und es nicht wahrnimmt. [104]

Fazit

Beide Systeme erfassen jeweils nur ein Symptom, das bei einem epileptischen Anfall auftritt. Der VitaGuard ist ein relativ großer, 700 Gramm schwerer Monitor, welcher mit den Sensoren zur Vitalfunktionsmessung ausgestattet ist. Daher ist dieser unhandlich und beeinflusst eventuell den Patienten beim Schlaf. Der Epi-Care 3000 hingegen misst Körperbewegungen und ist schwer einzurichten und laut Mitglieder von Foren sendet dieses Gerät bei manchen Patienten einige Fehlalarme und verfälscht somit die Werte. [105]

Des Weiteren verfügen beide Systeme über keine Netzwerkintegration, sodass sich Werte nicht automatisch an andere Systeme oder Server übermitteln lassen.

2.10.3 Richtlinie 93/42/EWG über Medizinprodukte

Diese Richtlinie ist eine von dreien, welche für medizinische Produkte aus dem deutschen und österreichischen Markt gelten und bildet das wichtigste Regelinstrument zur Gewährleistung von Sicherheit und der medizinisch-technischen Leistungsfähigkeit von Medizinprodukten. Diese müssen nach der Richtlinie 93/42/EWG alle Kriterien für die miteinander verbundenen Geräte, Instrumente, Vorrichtungen, Stoffe oder andere Gegenstände, welche für einwandfreies Funktionieren des Systems relevant sind, erfüllen, um als CE zertifiziertes Medizinprodukt auf den Markt zu kommen. Diese Richtlinie bezieht sich auf die Erkennung von Krankheitsbildern, die Überwachung und Untersuchung von Patienten. [106]

Das Amtsblatt der europäischen Gemeinschaft binhaltet die Anforderungen an Medizinprodukte, welche der Richtlinie 93/42/EWG entsprechen. Folglich wird ein kleiner Auszug der Anforderungen dargelegt.

- **Allgemeine Anforderungen**
  Die Anwendung darf weder den Zustand und die Sicherheit des Patienten, noch die des Anwenders oder Dritter gefährden. Bei der Auslegung und Konstruktion der Produkte müssen sich die Hersteller nach den Grundsätzen der integrierten Sicherheit richten. Diese umfassen die Beseitigung und Minderung von Risiken durch Sicherheitskonzepte bei der Herstellung und Konzeption des Produktes, Alarmvorrichtungen und Schutzmaßnahmen bei Risiken, welche durch die Entwicklung nicht beseitigt werden konnten und die Unterrichtung der Anwender über mögliche Risiken.

- **Anforderung an Auslegung und Konstruktion**
  Diese Anforderungen befassen sich mit den eingesetzten Werkstoffen, im speziellen mit toxischen und entflammbaren Materialien. Des Weiteren mit der Verträglichkeit zwischen verschieden eingesetzten Werkstoffen, dem Infektionsrisiko und der Anpassung an die Anwendungsumgebungsbedingungen. Bei Produkten mit Messfunktion müssen Genauigkeitsgrenzen entsprechend der Zweckbestimmung gewährleistet und ergonomische Grundsätze eingehalten werden. Ein weiterer wichtiger Punkt ist der Schutz vor Strahlung. Hierbei wird festgelegt, welche Arten von Strahlung vorkommen dürfen und wie darauf verwiesen werden muss. Generell gilt unbeabsichtigte Strahlung so weit es geht zu verringern. Ebenfalls sind Anforderungen an Produkte mit externer oder interner Energiequelle definiert, sodass die Gefahr von Störquellen für andere medizinische Geräte minimiert wird. [107]

2.10.4 ZigBee Standard

Dieser Standard wurde von der ZigBee-Allianz entwickelt und gilt als Industriestandard in Funknetzen, basierend auf dem Übertragungsprotokoll für »Wireless Personal Area Networks (WPAN)[63]«, welches die beiden unteren Schichten des OSI-Modells definiert. ZigBee bietet auf den höheren Protokollebenen Funktionen zum Routing wie auch eine Anwendungsschnittstelle an.

[63] Standard IEEE 802.15.4

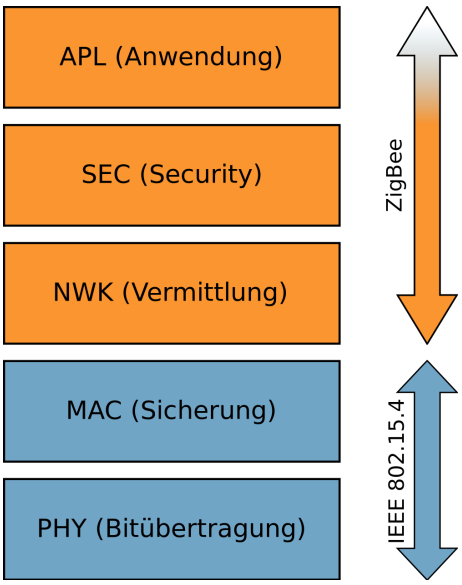


Abbildung 45: ZigBee-Stack [108]

Ziel war es, ein Protokoll zu entwickeln, welches durch geringe Leistungsaufnahme eine lange Batteriebetriebszeit gewährleistet, günstige Hardware, sichere Übertragung und Parallelbetrieb mehrerer Sender auf gleichen Frequenzen ermöglicht. Aufgrund dessen eignet sich der IEEE 802.15.4 Standard für drahtlose Sensornetze, bei denen die Sensoren am Körper getragen werden (WBAN – Wireless Body Area Network).
ZigBee wurde speziell dafür entwickelt, um Haushaltsgeräte, Sensoren und vieles mehr auf Kurzstrecken miteinander zu verbinden. [109]

IEEE 802.15.4 Netztopologien

Die einfachste Form einer Netzwerktopologie bildet die Punkt-zu-Punkt- oder Zweipunkt-Topologie, welche aus einem Koordinator und einem Knoten besteht.

Dieser Standard verfügt über eine Sterntopologie, bei der alle Endgeräte direkt mit einem sogenannten Koordinator kommunizieren. Der Koordinator ist meist ein leistungsfähiges Gerät, welches direkt an das Stromnetz angeschlossen ist, während die Endgeräte oftmals nur batteriebetrieben sind. Als solche Endgeräte können zum Beispiel flexibel einsetzbare Arduino-Boards fungieren, welche wiederum ihre Sensordaten an ein zentrales Arduino-Board senden, das als Koordinator zwischen den Boards agiert. [110]

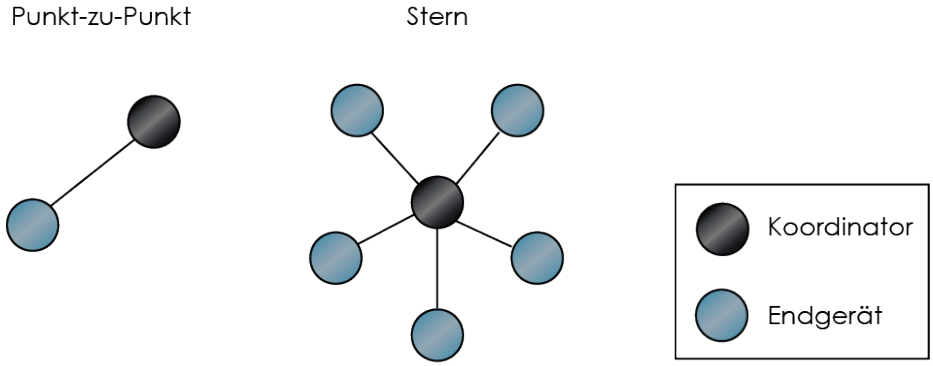


Abbildung 46: IEEE 802.15.4 Netztopologie

Durch ZigBee mögliche Topologien

Da der IEEE 802.15.4-Standard keine Vermittlungsschicht definiert, muss beispielsweise eine Funktion wie Routing von einem anderen Protokoll, welches auf einer höheren Schicht aufbaut, umgesetzt werden. Hier kommt der ZigBee Standard zum Einsatz und übernimmt das Routing und ermöglicht somit weitere Toplogien. Daher steht zum Beispiel die Baumstruktur-Topologie zur Verfügung, eine Mischung aus den beiden oben genannten, wobei es sich allerdings um keine vollständige Vermarschung handelt. Das Rückgrat dieser Topologie bildet wieder der Koordinator, welcher wiederum mit Routern und Endgeräten verbunden sein kann und somit Daten von einem Endgerät direkt oder über einen Router an den Koordinator übermittelt.

Ein vermarschtes Netz gilt als sicherste Variante unter den Topologien, da falls ein Knoten ausfallen sollte, jeder andere die Funktionalität übernehmen und die Daten oder Informationen weitervermitteln kann. Hierbei werden sogenannte Router-Knoten eingesetzt, welche wiederum an dem Koordinator oder an einem anderen Router angebunden sein können. Es ist quasi eine erweiterte Topologie der Baumstruktur. [110]

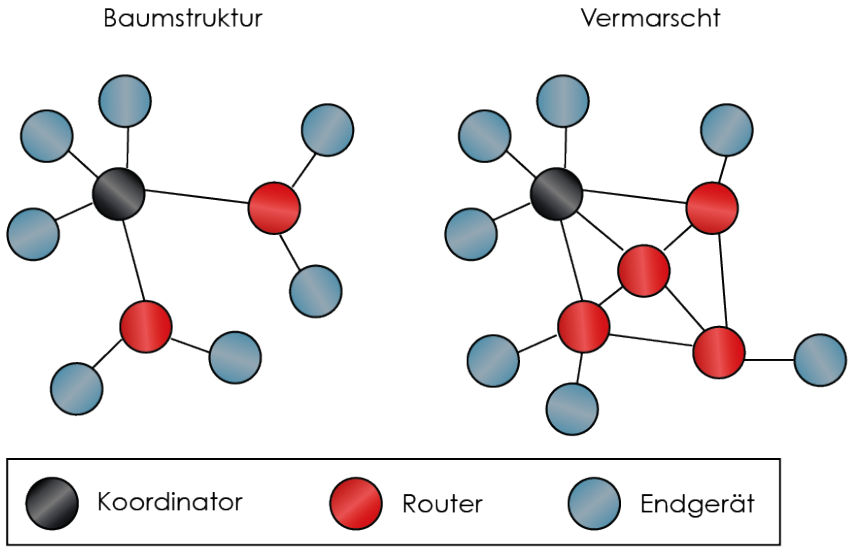


Abbildung 47: ZigBee erweiterte Netztopologie

3 ANFORDERUNGSANALYSE

In diesem Kapitel werden konkrete Anforderungen an ein Arduino-System im Bereich Monitoring im Gesundheitswesen formuliert. Hierzu ist es wichtig die Aspekte, welche auf Anwenderseite gegenüber einem Produkt in diesem Bereich relevant sind, herauszufinden, auf Umsetzbarkeit zu überprüfen und die Relevanz in Bezug auf Implementierung eines Prototyps zu beurteilen. Dazu werden die Aspekte in funktionale wie auch nicht-funktionale Anforderungen eingestuft und anschließend priorisiert, um den Umfang und die Funktionalität des Prototyps definieren zu können.

3.1 Projekt-Scope

Die Erhebung und Analyse der Anforderungen befasst sich mit einer Monitoring-Anwendung im Bereich der Epilepsie in der Schlafphase. Genauer befasst sich das Projekt mit der in Kapitel 2.10.1 beschrieben Anfallsform (tonisch-klonischer Anfall), welche Symptome wie Verkrampfung der Muskulatur und dadurch hervorgerufene Muskelzuckungen der Gliedmaßen wie auch die Sauerstoffunterversorgung und den daraus resultierenden erhöhten Puls umfassen.

Es soll dahingehend ein Konzept für einen Prototyp im Anschluss dieses Kapitels entwickelt werden, welches in einer vernetzten Umgebung Daten von Sensoren zu Messung dieser Symptome an einen Server sendet und somit die Überwachung des Patienten und die Analyse der im Schlaf gewonnenen Daten, gewährleistet. Eine Weiterverarbeitung der anschließend auf dem Server archievierten Daten ist in diesem Umfang nicht relevant, sondern ausschließlich die Fähigkeit des Arduinos als Schnittstelle zwischen physischer und virtueller Welt im Bereich einer medizinischen Anwendung zu erläutern und aufzuzeigen. Dennoch werden Anforderungen an ein solches System definiert, da Arduino als Produktentwicklungstool, als ein Werkzeug zur Entwicklung und Umsetzung eines marktreifen Produktes, fungieren kann und somit nach einer Prototypentwicklungsphase die gewonnenen Daten und Informationen für die Weiterentwicklung verwendet werden können.

Über den Umfang der Bachelorarbeit hinaus soll als finales angedachtes Produkt ein Armband für unter Epilepsie leidende Kinder entwickelt werden. Dieses soll auf der medizinisch technischen Grundlage den Richtlinien 93/42/EWG für Medizinprodukte entsprechen und unter einem für Kinder ansprechenden Designaspekt gestaltet werden, um somit dem betroffenen Patienten das Gefühl einer medizinischen Untersuchung beziehungsweise Überwachung zu nehmen. Hierbei ist von großer Relevanz, die Hardwarekomponenten kleinstmöglich zu halten und die Verkabelung der Sensoren und Stromversorgung innerhalb des Armbandes durchzuführen und keine offenen, sichtbaren Leiterwege aufzuweisen.

3.2 Funktionale Anforderungserhebung

In dieser Erhebung werden die Anforderungen an die von einem System erwarteten Funktionalitäten ermittelt. Diese umfassen insbesondere die systeminternen Funktionen wie auch die Interaktion durch den Anwender.

3.2.1 Eingabe/Erfassung

Für ein Projekt in diesem Bereich sind zwei Sensoren erforderlich, welche die gewünschten zu untersuchenden Daten liefern. Zum einen wird auf einen Puls-Oxigen-Sensor zum Messen von Sauerstoffsättigung und Herzfrequenz zurückgegriffen, zum anderen auf einen Accelerometer zum Bestimmen von Bewegungen beziehungsweise Muskelzuckungen. Diese beiden Faktoren umschließen die Symptome, welche bisher jeweils nur einzeln durch ein auf dem Markt erhältliches System erfasst werden können. Durch ein derartiges System kann in Abhängigkeit der beiden Parameter die Wahrscheinlichkeit von Fehlalarmen minimiert werden, da eine Erhöhung des Pulses nicht zum Auslösen des Alarms genügt. Es müssen beide Sensoren erhöhte Aktivitäten oder Werte feststellen, um einen Anfall als solchen zu bestimmen.

Pulse-Oxigen-Sensor

Diese Sensoren dienen der nicht invasiven[64] Ermittlung arterieller Sauerstoffsättigung und Herzfrequenz durch Lichtabsorption bei Durchleuchtung der Haut. Um genauere Werte bei der Messung des Sauerstoffgehalts im Blut zu erlangen, ist es notwendig, dass der Sensor neben der Leuchtdiode mit rotem Licht (für gewöhnlich einer Wellenlänge von 660nm) zusätzlich über eine Infrarot-Diode (Wellenlänge zirka 910nm) verfügt, um so aus dem Verhältnis der beiden Lichtabsorptionen den Sauerstoffgehalt zu ermitteln. Ansonsten kann durch stärkere Gewebeschichten oder sonstigen Faktoren das Ergebnis verfälscht werden. Dahingehend ist eine Kombination aus beiden Lichtquellen für den medizinischen Bereich beziehungsweise zur genauen Messung von Vitalfunktionen unabdingbar.

Accelerometer

Der Accelerometer oder auch Beschleunigungssensor soll zum Messen von Bewegungen beziehungsweise von Muskelzuckungen eingesetzt werden. Hierbei ist es wichtig, dass alle drei Achsen (x, y und z) gemessen werden können. Nur durch derartige Sensoren ist es, unabhängig von der Lage des Armes und der damit verbundenen Ausrichtung des Sensors möglich, jede Bewegung beziehungsweise Zuckung festzustellen. Durch das Erfassen der Beschleunigung kann somit auch von einer im Normalschlaf ausgeübten Bewegung unterschieden werden und somit ein epileptischer Anfall eindeutig festgestellt werden.

Zusammenfassung

Nur durch beide Sensoren kann eindeutig ein Anfall erfasst werden. Der Herzschlag wie auch die Sauerstoffsättigung geben genügend Informationen für die Auswertung für Ärzte bezüglich der Schwere des Anfalls. Allerdings kann durch den Bewegungssensor zusätzlich Auskunft über den genauen Verlauf der einzelnen Phasen und der Symptome geliefert werden und dieser zur Vermeidung von Fehlalarmen eingesetzt werden.

3.2.2 Verarbeitung

Die erfassten Werte durch den Puls-Oxigen-Sensor sollen mit den festgelegten Vitalwerten des Patienten (Ruhepuls/Sauerstoffsättigung) abgeglichen werden, um somit eine Erhöhung der Herzfrequenz wie auch Verringerung der Sauerstoffsättigung feststellen zu

[64] gewebeverletzenden

können. Die gewonnenen Daten müssen einerseits für den sofortigen Upload auf einem Server bereitgestellt werden, wie auch in eine Variable oder in einen Array zwischengespeichert werden, um somit eine kurzzeitige Historie über die letzten Minuten zu erlangen. Nach einem bestimmten durch den Benutzer festgelegten Zeitintervall in Bezug auf erhöhten Puls und unter Normalwert definierter Sauerstoffsättigung soll in Abhängigkeit zu den erfassten Daten des Accelerometers ein Alarm ausgelöst werden. Dies bedeutet, dass einerseits die Daten der beiden Sensoren kontinuierlich auf den Server übertragen werden müssen, andererseits auch ab Beginn einer der Symptome das System in eine zeitbezogene Überwachungsphase übergeht. Wenn hierbei der Anfall eine definiert lange Zeit anhält, muss eine Funktion zur Alarmierung ausgeführt werden.

Des Weiteren muss zur Übertragung der Werte an einen Server das System eine Verbindung im lokalen Netzwerk zu einem externen Server aufbauen können, da die Werte für den behandelnden Arzt zur Auswertung und zur Dosierung der Medikation relevant sind.

3.2.3 Ausgabe

Die Ausgabe ist hauptsächlich für den jeweiligen behandelnden Arzt erforderlich. Diesbezüglich sollen in Zeit/Werte Diagrammen der tägliche, wöchentliche, monatliche und jährliche Verlauf von Herzfrequenz, Sauerstoffsättigung und Bewegungsabläufe angezeigt werden. Des Weiteren muss gewährleitstet sein, dass zu den jeweiligen Zeitpunkten der Messung auch der Wert von Sauerstoffsättigung und Herzfrequenz angezeigt werden kann. Bei der Darstellung der Bewegungsabläufe ist es nicht von Nöten nähere Informationen auszugeben, hier genügt lediglich die Visualisierung der Bewegungsabläufe. Dies kann wie folgt aussehen.

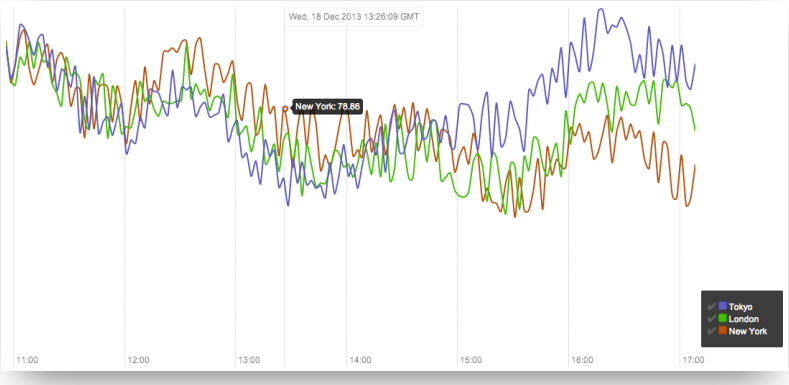


Abbildung 48: Liniendiagramm Zeit/Wert-Verlauf [111]

Für den behandelnden Arzt soll hierzu eine Plattform mit entsprechenden Sicherheits-Protokollen zum Schutz der Patientendaten zur Verfügung stehen, auf der dieser sich einloggen und seine Patienten verwalten kann. Die Armbänder werden mit einem Identifier ausgeliefert, welchen der Arzt in die Plattform mitsamt den Patientendaten einzugeben hat und somit aktiviert und dem Patienten zuordnet. Die Armbänder senden ihren Bezeichner beim Übermitteln der Daten an den Server mit, sodass diese in einer sicheren Datenbank abgelegt und gespeichert werden können.

Sind die Werte nach einer definierten Zeit noch immer über den Normalwerten, so soll ein Alarm beziehungsweise Warnsignal ausgesandt werden. Dies dient der Alarmierung von Eltern oder Pflegeperson, bevor irreversible Schäden beim Patienten auftreten können. Dieses Signal kann in Form eines Anrufes an einer hinterlegten Telefonnummer umgesetzt werden.

3.2.4 Speicherung

Die erfassten Werte sollen für Langzeitanalysen und zur Überwachung gespeichert werden. Hierfür ist es wichtig, die Informationen in einer Datenbank auf einem Server abzulegen, welche gegebenenfalls kontinuierlich gesichert wird, um einen Verlust dieser Daten zu vermeiden. Da das System über einen Router im lokalen Netzwerk ins Internet kommuniziert, ist es sinnvoll aufgrund dieser systemexternen Kollisionsdomäne eine lokale Speicherung auf dem System durchzuführen, um bei Verlust oder Zusammenbruch einer Verbindung über gesicherte lokale Daten zu verfügen.

Des Weiteren sollten auf dem System die Konfigurationsparameter für den Netzwerkverbindungsaufbau gespeichert werden, damit nicht nach jedem Start diese neu beschrieben werden müssen.

3.2.5 Systeminteraktion

Dem Anwender dieses Systems müssen Schnittstellen zur Verfügung stehen, um die Parameter für Ruhepuls und Sauerstoffsättigung einstellen zu können. Des Weiteren soll es möglich sein, dass bei Kindern mit erhöhten Bewegungsradien und Aktionen im Schlaf, der Accelerometer entsprechend eingestellt beziehungsweise fein justiert werden kann. Unter anderem muss für den Anwender im Anwendungsfall einer WLAN-Lösung die Eingabe der Netzwerkparameter wie Bezeichner des Netzwerks und zugehöriges Passwort möglich sein.

3.2.6 Funktionale Vereinbarung

Der Anwender ist für den Kommunikationsaufbau zuständig und muss das System über einen Computer konfigurieren können. Hierzu müssen unter der Verwendung eines WLAN-Moduls die Verbindungsparameter wie Name und Passwort eingegeben werden, die Telefonnummer für das Warnsignal wie auch die Normalwerte des Patienten, die im günstigsten Fall vom behandelten Arzt gemessen werden.

3.3 Nicht-Funktionale Anforderungserhebung

Diese Anforderungen umfassen die Randbedingungen des Systems und gliedern sich in Produkt- und Prozess-Anforderungen und behandeln die Benutzbarkeit, Qualität und die Art, wie das System zu arbeiten hat.

3.3.1 Technische Anforderungen

Das System im Protypenstadium muss mit Arduino entwickelt werden und als Standalone-Anwendung funktionieren, sodass nach Einrichten der für den Betrieb notwendigen Funktionen, das Produkt an das Netz angeschlossen werden kann und für einen Dauerbetrieb in dem eingerichteten Netzwerk ausgelegt ist, ohne dass weitere Konfigurationen vorgenommen werden müssen.

Nach dem Prototypenstadium kann ein eigener Schaltkreis entwickelt werden, welcher spezifisch auf die Anwendung ausgelegt ist und lediglich als Logikeinheit den Atmel-Mikrocontroller umfasst. Dadurch ist eine Minimierung des Systems möglich. Zudem werden zum Erfassen der Sensordaten analoge wie auch digitale Eingänge benötigt.

3.3.2 Ergonomie

Hierbei wird in zwei Kategorien unterschieden. Zum einen muss das System die gewonnenen Daten in einem für Parser einfach und schnell zu lesenden Format zur Verfügung stellen. Zum anderen setzt sich diese Kategorie auch mit den ergonomischen Eigenschaften des Produktes auseinander. Dahingehend muss in dem Anwendungsumfeld, Überwachung von Funktionen bei Kindern im Schlaf ein hoher Tragekomfort gewährleistet sein, sodass das System ohne Einschränkungen und Beeinträchtigungen im Schlaf getragen werden kann. Hierbei soll auf äußerliche medizinische Gestaltung verzichtet werden, damit Kinder dieses eher als Schmuck anstatt als System zum Überwachen ihrer Vitalfunktionen betrachten. Des Weiteren müssen die Hardwarekomponenten kleinstmöglich und mit möglichst wenig Gewicht konzipiert werden. Kabel sollten hierbei nur von den Sensoren zum Mikrocontroller geführt und nicht für die Versorgung für Strom oder externe Displays verwendet werden, was wiederum die Bewegungsfreiheit und den Komfort beeinflussen würde.

3.3.3 Bedienbarkeit

Durch eine Software soll es dem Anwender möglich sein die Parameter für die Netzwerkverbindung und der Vitalwerte einzugeben. Diese muss plattformunabhängig und einfach zu bedienen sein, sodass eine auch für Laien verständliche Konfiguration möglich ist. Das Anbringen des Armbandes und der Sensoren muss durch eine Bedienungsanleitung erläutert werden, damit der Puls-Oxigen-Sensor richtig arbeiten kann und keine fehlerhaften Werte liefert. Des Weiteren muss der Accelerometer justierbar sein, dass dieser an die spezifischen Eigenschaften des Patienten angepasst werden kann.

3.3.4 Leistung

Der Mikrocontroller muss über einen Speicher von mindestens 30 KB verfügen, damit der kompilierte Sketch darauf gespeichert werden kann. Das System muss zudem fähig sein, den ZigBee-Standard zu unterstützten und über ein WLAN- oder Ethernet-Modul mit dem Router zu kommunizieren. Hierbei soll über zwei separate Moduleinheiten gearbeitet werden. Das in dem Armband integrierte Modul ist dabei zum Erfassen der gemessenen

Daten und für das Weiterleiten dieser an ein weiteres Modul über den ZigBee-Standard zuständig. Das zweite System fungiert hierbei als Schnittstelle ins Internet und bildet der Gateway zwischen dem ZigBee-Standard und dem TCP-IP Protokoll.

Der ZigBee-Standard soll hierbei eingesetzt werden, da dieser wesentlich weniger Energie als WLAN-Module benötigt und die Module über eine Schlaf- und Aufweck-Funktion verfügen, sodass bei Nichtgebrauch die Kommunikation eingestellt und der Stromverbrauch gesenkt wird. Dadurch bleibt das System über Monate hinweg einsatzfähig ohne einen Batteriewechsel oder Ladezyklus vorzunehmen.

3.3.5 Messgenauigkeit

Für die Auswertung in medizinischer Hinsicht ist eine genaue Messung der Herzfrequenz und des Sauerstoffgehalts unabdingbar und von hoher Relevanz. Übliche Toleranzen liegen hierbei bei der Sauerstoffsättigung bei ± 3 % und bei der Herzfrequenz bei ± 2 bpm[65]. Die Bewegungen beziehungsweise die Stärke der Muskelzuckungen ist dabei irrelevant, wichtig ist hierbei nur, zu sehen, wann diese einsetzen und wie lange die Phasen anhalten.

3.3.6 Energieversorgung

In diesem Bereich müssen die beiden Systeme wiederum separat betrachtet werden. Das Armband muss über eine kleine, leistungsstarke Batterie verfügen, sodass eine lange Betriebslaufzeit erzielt werden kann. Durch energieeffiziente Komponenten kann ebenfalls der Verbrauch gesenkt und somit eine längere Laufzeit erzielt werden.

Das Modul für die Netzwerkkommunikation ins Internet kann über das Stromnetz mit Energie versorgt werden, da es an einen Standort angebracht und nicht bewegt werden muss.

3.3.7 Zuverlässigkeit

Das System muss absolut zuverlässig sein, sodass keine Fehlalarme ausgelöst und bei Dauerbetrieb das Versagen von Hardwarekomponenten minimal gehalten werden. Daher ist es von Nöten, dass der Anwender darauf hingewiesen wird, Stichproben der erfassten Informationen zu nehmen, indem dieser auf der Webseite oder Plattform, auf der die Daten letztendlich dargestellt werden, diese abruft und überprüfen und gegebenenfalls bei Auffälligkeiten das System neu konfigurieren oder justieren kann.

Durch Warnhinweise bei niedrigem Energiestand der Batterie soll über akustische und visuelle Signale auf einen baldigen Austausch der Energiequelle hingewiesen werden. Ein Warnhinweis ist ebenfalls für zu hohe oder niedrige Messwerte (derart hohe Abweichung, dass Fehlfunktion vorliegt) beim Anbringen des Armbandes bedeutsam, sodass zeitnah für den Anwender und Patienten ersichtlich wird, ob das Modul oder der Puls-Oxigen-Sensor fachgerecht angebracht wurde.

[65] »Beats per minute« zu deutsch, Schläge pro Minute

3.3.8 Zulassung gemäß Richtlinie 93/42/EWG

Das Produkt kann, muss allerdings nicht, den Richtlinien gemäß 93/42/EWG für Medizinprodukte entsprechen. In dem angestrebten Fall, dass dieses Produkt über Ärzte verschrieben und die Kosten von Krankenkassen übernommen werden, ist das Einhalten dieser Richtlinien hingegen unabdingbar und somit erforderlich.

3.4 Analyse und Priorisierung der Anforderungen

In der folgenden Analyse werden die oben genannten Anforderungen in Bezug auf die Arduino-Plattform und auf Realisierbarkeit überprüft. Des Weiteren werden Prioritäten bezüglich des anschließenden Konzeptes für einen Prototyp vergeben. Hierbei gilt der Wert »0« für nicht relevante Anforderungen für den Prototyp, »1« für Features und »2« für die Einbindung und Umsetzung im Prototyp.

3.4.1 Eingabe/Erfassung

Daten und Informationen über Vitalfunktionen und Bewegungsabläufe des Patienten müssen mittels Sensoren erfasst werden.

- Für Arduino stehen diverse Accelerometer wie auch gehackte oder für Arduino entwickelte Puls-Oxigen-Sensoren mit entsprechender Bibliothek zur Verfügung.

  - **Priorität 2:**
    Beide Sensoren sind für den Prototyp aufgrund der Erfassung der zu überwachenden Daten unabdingbar und sind aufgrund der auf dem Markt erhältlichen Komponenten auch prototypisch implementierbar.

3.4.2 Verarbeitung

Die Verarbeitung wie auch die Kommunikation mit dem Router findet im günstigsten Fall in einem System statt. Das Armband soll hierbei die Daten erfassen, in das richtige Format bringen und anschließend über den ZigBee-Standard an einem am Router angeschlossenen ZigBee zu TCP-IP Gateway senden.

- Der Hersteller der XBee-Module für den ZigBee-Standard für Arduino-Boards namens »Digi« hat seit Mitte des Jahres 2013 sein Gateway aus der Produktpalette für den europäischen Markt genommen. Aufgrund dessen muss über ein zweites Arduino-Board, erweitert mit WiFi- oder Ethernet-Shield, ein entsprechendes Gateway eigenständig implementiert werden. Dieses System wird im weiteren Verlauf als Client bezeichnet, da dieses für die Kommunikation ins Internet zuständig ist und die eingehenden Daten per HTTP-Request übermittelt.
- Das zweite System bildet ein Arduino-Board mit XBee-Modul und angeschlossenen Sensoren. Dieses wird im weiteren Verlauf als Produzent

bezeichnet, da dieses ausschließlich für die Gewinnung der Daten und die Weiterleitung an das nächste Arduino-System zuständig ist.

- **Priorität 2:**
  Die Verarbeitung und Vermittlung innerhalb des Home-Area-Netzwerkes gilt prototypisch zu implementieren aufgrund der relevanten Rolle als Niedrigenergie-Standard des ZigBee oder IEEE 802.15.2 in diesem System. Das Arduino-Board verfügt über eine CE-Zertifizierung für den Schaltkreis, allerdings nicht in Verbindung mit Funkmodulen. Insofern ist dieses System ohne weitere Ausstellung einer Zertifizierung nicht CE-zertifiziert[66]. Zudem kann die Verbindung ZigBee oder IEEE 802.15.4 und Arduino-Board nicht so gesteuert werden, dass wenn keine Daten erfasst werden, das gesamte System in einen Ruhezustand versetzt wird und somit weniger Energie verbraucht.

3.4.3 Ausgabe

Die Ausgabe der Daten findet über eine Plattform im Internet statt. Diese verfügt über Sicherheits-Protokolle, eine Datenbank wie auch eine Eingabemaske für die Patientendaten. In einem Zeit/Werte-Diagramm werden die gewonnenen Daten visualisiert. Des Weiteren soll eine Warnfunktion hinzugefügt werden, worauf die Betreuer des Patienten über einen zu lang andauernden Anfall hingewiesen werden.

- Aufgrund der serverseitigen Konfiguration und Entwicklung, ist dieses System (Webseite/Datenbank) unabhängig von Arduino. Relevant hierbei ist nur die Formatierung der gewonnenen Daten für den Parser auf dem Server.

  - **Priorität 1:**
    Die Ausgabe über ein Patientenportal ist nicht im Zuge der Entwicklung eines Prototypen relevant und wird dahingehend auch nur über einen PaaS-Service für die Visualisierung von Sensordaten realisiert.

- Ein System mit Warnfunktion kann mittels GSM-Shield für Arduino-Boards realisiert werden. Hierzu wird zusätzlich eine SIM-Karte benötigt, über die ein Anruf getätigt werden kann. Der Anruf Bedarf keiner Audio- beziehungsweise Ton-Übertragung, da lediglich das Klingeln des Empfänger-Telefons als Warnsignal ausreicht.

  - **Priorität 1:**
    Diese Funktion ist zwar möglich aufgrund der erhältlichen Hardware und Erweiterungen, allerdings kann eine Warnfunktion auch mittels Push-Nachricht oder E-Mail umgesetzt werden. In Bezug auf einen Prototyp gilt diese Funktion allerdings als Feature und wird nicht umgesetzt. Eine Implementierung allerdings ist möglich.

[66] Eine CE-Zertifizierung kann eigenständig erstellt werden und gibt keinerlei Auskunft über die Qualität des Produktes oder Systems.

3.4.4 Speicherung

Es wird eine Speicherung auf einer Datenbank für die Visualisierung der Daten im Webbrowser, eine lokale Speicherung auf einer SD-Karte und Speicherung von Konfigurationsparametern für den Verbindungsaufbau unter der Verwendung eines WiFi-Shields vorgesehen.

- Die Speicherung auf der Datenbank findet wie unter Kapitel 3.4.3 serverseitig statt.

  - **Priorität 1:**
    Diese Funktionalität ist realisierbar, allerdings für die Prototyp-Entwicklung nicht von Bedeutung. In diesem Stadium kann wiederum auf einen PaaS-Service zurückgegriffen werden, welcher die Daten in einer eigenen Datenbank speichert, allerdings keine Patientenverwaltung umfasst.

- Eine lokale Speicherung der Daten auf einer SD-Karte ist über ein WiFi- oder Ethernet-Shield möglich, welches über einen SD-Slot verfügt. Des Weiteren sind auf dem Markt spezielle SD-Shields vorhanden. Eine direkte Speicherung auf einem Computer im Heimnetzwerk kommt nicht infrage, da ein laufendes System (Computer) hierzu erforderlich wäre.

  - **Priorität 0:**
    Lokale Datenspeicherung ist für die Prototyp-Entwicklung von keiner großen Bedeutung, da dies ausschließlich eine Sicherheitsmaßnahme vor Datenverlust umfasst.

- Das Speichern von Konfigurationsparametern kann über den EEPROM-Speicher vorgenommen werden. Dieser kann auch über das Netzwerk beschrieben werden und muss somit nicht an den Computer angeschlossen werden.

  - **Priorität 1:**
    Im Prototyp sollen lediglich im Programmcode die Werte eingestellt beziehungsweise die Konfigurationsparameter beschrieben werden. Für eine Anwendung im Auslieferungsstadium hingegen wäre eine Software nötig, sodass auch Computer-Laien die Werte festlegen können.

3.4.5 Systeminteraktion

Dem Anwender muss eine Schnittstelle zum System zur Verfügung stehen, über diese der Anwender die Parametrierung der Werte für Sauerstoffsättigung, Puls und die Sensibilität des Beschleunigungssensors vornehmen kann. Zusätzlich müssen bei einem WLAN-System die Netzwerkparameter eingegeben werden, um eine Verbindung zum Router herstellen zu können.

- **Priorität 0:**
  Der Arduino-UNO verfügt über nur eine serielle Schnittstelle, welche allerdings für die Kommunikation im PAN-Netzwerk benötigt wird. Eine Weitere serielle Schnittstelle kann allerdings softwarebasierend hinzugefügt werden. Für den Prototyp ist allerdings keine Alarmfunktion vorgesehen, sodass die Eingabe von Normalwerten für Puls und Sauerstoffsättigung nicht als Referenzwert benötigt wird. Die Parametrierung der Sensibilität des Beschleunigungssensors wird direkt im Programmcode vorgenommen und prototypisch auf einen vom Autor gewählten Wert festgelegt.

3.4.6 Funktionale Vereinbarung

Die Konfiguration des Systems muss durch den Anwender vorgenommen werden.

- **Priorität 0:**
  Diese Anforderung bezieht sich rein auf die Anwenderseite und ist daher für den Prototyp nicht relevant.

3.4.7 Technische Anforderungen

Dieser Punkt umfasst die Anforderung an eine Standalone-Anwendung, welche nach Programmierung selbstständig arbeitet und keine weitere Interaktion durch den Anwender bedarf, um funktionsfähig zu arbeiten. Des Weiteren werden mehrere analoge wie auch digitale Eingänge benötigt, über welche die Sensordaten auf das Board gelangen und die Erweiterungen angeschlossen werden.

- Arduino-Boards speichern den Programmcode dauerhaft, sodass auch nach Entfernen der Stromversorgung der Programmcode erhalten bleibt.

  - **Priorität 2:**
    Nach der Programmierung und Fertigstellung des Prototypen soll das System innerhalb des konfigurierten Netzwerkes funktionsfähig sein, unabhängig davon wie oft das System neu gestartet wird.

- Jedes Arduino-Board verfügt über mehrere analoge und digitale Eingänge. Des Weiteren kann auch über die digitalen PWM-Eingänge ein analoges Signal simuliert werden, allerdings mit einer geringeren Genauigkeit.

  - **Priorität 2:**
    Diese Eingänge werden zum Erfassen der Sensordaten benötigt. Die genaue Anzahl der benötigten Eingänge wird im Verlauf der Implementierung ermittelt.

3.4.8 Ergonomie der Daten

Die Daten müssen für die serverseitigen Parser lesbar formatiert werden.

- Im Programmcode können die Daten mit oder ohne Bibliotheken in diverse Formate formatiert werden. Häufig werden diese in JSON-Objekte geschrieben, wozu auch eine Bibliothek namens »aJson« zur Verfügung steht.

  - **Priorität 2:**
    Die Daten müssen in ein für die API des PaaS-Services lesbares Format gebracht werden.

3.4.9 Ergonomische Eigenschaften

Ergonomie setzt sich mit den Eigenschaften wie Tragekomfort des Armbandes und Verkabelung wie auch Design des Produktes auseinander.

- Arduino bietet in seiner Produktpalette auch tragbare Boards an, welche in Textilien eingenäht werden können. Aufgrund der Flexibilität des Boards und der geringen Größe würde sich dieses für einen derartigen Anwendungsfall eignen.

  - **Priorität 0:**
    Diese Eigenschaften sind in der Phase bei der Entwicklung irrelevant, da lediglich auf den Funktionalitätsumfang eingegangen wird. Die dadurch zu verwendeten Erweiterungs-Shields sind darüber hinaus meist für den Arduino-UNO verfügbar, weshalb sich dieses Board zur Entwicklung anbietet. Wichtig ist nur zu wissen, dass derartige Boards existieren und in einer späteren Entwicklungsphase darauf zurückgegriffen werden kann.

3.4.10 Bedienbarkeit

Diese Anforderung setzt sich rein mit der Anwenderseite auseinander und umfasst die Usability der Software zur Konfiguration des Systems wie auch eine Bedienungsanleitung für fachgerechte Anwendung.

  - **Priotität 0:**
    Die Usability und eine Bedienungsanleitung sind für die Prototypen-Entwicklung irrelevant, welche rein die Funktionalität der Hardwarekomponenten umfasst. Eine Softwaretechnische-Lösung zum Konfigurieren des Systems ist möglich, allerdings nicht Bestandteil dieser Thesis.

3.4.11 Leistung

Es muss genügend Speicherkapazität für den Programmcode, Bibliotheken und Parameter zur Verfügung stehen. Der ZigBee- oder IEEE 802.15.4-Standard muss unterstützt und eine Kommunikation mit dem Internet gewährleistet sein. Des Weiteren soll eine lange Betriebszeit des Armbandes möglich sein.

- Arduino bietet verschiedene Boards mit unterschiedlicher Leistung an, darunter auch der Arduino-UNO mit 32 KB Flash-Speicher, was für die Anwendung als Armband-Modul wie auch Gateway ausreichen müsste.

  - **Priorität 2:**
    Das Board muss, um den Programmcode ausführen zu können, diesen auch komplett speichern können.

- Wie in Kapitel 3.4.2 erläutert, wird vom Hersteller »Digi« ein Arduino-kompatibles Shield wie auch zugehöriges ZigBee- oder IEEE 802.15.4-Modul angeboten.

  - **Priorität 2:**
    Eine Kommunikation über den ZigBee- oder IEEE 802.15.4-Standard ist für den Prototyp relevant, da dieser Standard auch bei einem marktreifen Produkt eingesetzt werden kann. Allerdings lassen sich in Verbindung mit einem Arduino-Board Funktionen ausgehend von einem ZigBee-Modul wie Ruhezustand und vieles mehr nicht realisieren. Da für das Produkt ebenfalls nur eine Punkt-zu-Punkt-Verbindung ausreicht und nicht mit weiteren Sensorknoten kommuniziert werden muss, ist der IEEE 802.15.4-Standard hierfür absolut ausreichend.

3.4.12 Messgenauigkeit

Sensoren zur Messung von Vitalfunktionen haben einen gewissen Toleranzbereich. Dieser ist bei der Auswahl der Komponenten zu beachten.

- Diese ist keine Arduino spezifische Anforderung, da die Toleranzen am Sensor anliegen und die Abweichungen nicht vom Arduino verursacht werden. Allerdings kann durch eine gute Wahl der Sensoren die Messgenauigkeit beziehungsweise der Toleranzbereich bestimmt und festlegt werden. Die Messgenauigkeit ist lediglich beim Puls-Oxigen-Sensor relevant.

  - **Priorität 1:**
    Bei der Wahl der einzusetzenden Komponenten soll auf eine gewisse Genauigkeit geachtet werden, allerdings lassen sich verschiedene Sensoren unterschiedlich gut ansprechen und daher wird beim Prototyp versucht, einen leicht anzusprechenden (am besten durch eine Bibliothek) und dennoch messgenauen Sensor zu finden.

3.4.13 Energieversorgung

Beim Armband soll eine kleiner Lithiumionenakku und beim Gateway eine 240 V Versorgung über das Hausstromnetz gewährleistet sein.

- Arduino-Boards können von Batterien mit 7 V bis 12 V Spannung versorgt werden. Auf dem Markt gibt es ausreichend Lithiumionenakkus mit der gewünschten Spannung. Über ein Netzteil mit Barrel-Jack-Stecker kann das Arduino-Board, welches als Gateway fungiert mit 7 V bis 12 V konstant über das Stromnetz versorgt werden.

  - **Priorität 0:**
    Für den Prototyp ist es ausreichend, dass das Arduino-Board mit einer Batterie versorgt werden kann. Hierbei kann daher auch auf einen 9 V Batterieblock zurückgegriffen werden.

3.4.14 Zuverlässigkeit

Das System muss in diesem Anwendungsgebiet äußerst zuverlässig arbeiten und muss durch Sicherheitsfunktionen vor Ausfall und Störungen gesichert werden.

- Visuelle wie auch akustische Warnsignale können mit einem Arduino-Board umgesetzt werden. Des Weiteren können durch den Programmcode sogenannte Sicherheitsmechanismen eingebaut werden, welche kontinuierlich das System bezüglich Fehlmessungen oder zu geringem Energiestand messen und gegebenenfalls ein Alarmsignal aussenden. Durch eine Schleife im Programmcode kann beim Anlegen des Armbandes auch überprüft werden, welche Messwerte momentan anliegen, in dem zum Beispiel ein Mittelwert aus den 10 ersten Messungen genommen wird, welcher dann wiederum als SMS oder E-Mail an den Betreuer gesendet werden kann. Somit ist dieser in der Lage, die Anbringung des Armbandes zu überprüfen.

  - **Priorität 0:**
    Diese Maßnahmen, um einen hohen Grad an Zuverlässigkeit zu gewährleisten, ist in diesem Stadium der Entwicklung noch nicht von Bedeutung.

3.4.15 Zulassung gemäß Richtlinie 93/42/EWG

Systeme und Produkte, welche diese Richtlinien einhalten, können von Ärzten verschrieben und die Kosten teilweise oder ganz von Krankenkassen übernommen werden. Diese Richtlinien stellen wiederum Anforderungen an das Produkt, welche über Beschaffenheit des verarbeiteten Materials, Sicherheitsrisiken, Strahlungen und vieles mehr bestimmen.

- Ein Arduino-Board eignet sich nicht für marktreife Produkte im medizinischen Bereich. Allerdings ist es zur Entwicklung dieser geeignet und kann als

Prototyp eingesetzt werden. Nach einer erfolgreichen Testphase kann ein entsprechender Schaltkreis aufgebaut und entwickelt werden, welcher lediglich über den programmierten Atmel-Chip verfügt und der restliche Aufbau gemäß den Richtlinien erfolgt.

- **Priorität 0:**
  Eine Implementierung gemäß dieser Anforderung ist für die Prototypen-Phase nicht relevant. Daher müssen diese Richtlinien anschließend in Betracht bezogen werden, was allerdings nicht mehr Bestandteil der Thesis ist.

3.5 Use-Case Diagramm

Das Use-Case Diagramm zeigt das Leistungsspektrum des Systems auf, wie auch das Verhalten aus Sicht des Anwenders. In diesem Diagramm gibt es zwei Akteure, zum einen den Anwender der Arduino-Anwendung und zum anderen den behandelnden Arzt, der die Daten aus dem Internet über seinen Web-Browser abruft.

Der Anwender muss hierzu einerseits das System aktivieren, wie auch einige Konfigurationen vornehmen. Das »System aktivieren«-Use-Case stellt weitere Beziehungen zu anderen Use-Cases her. So werden nach der Aktivierung durch den Anwender automatisch Sensorwerte erfasst, worunter die Messungen der Sauerstoffsättigung wie auch der Puls fallen. Des Weiteren werden die Bewegungsabläufe gemessen. Die gesammelten Sensorwerte wiederum werden an einen Server übermittelt und anschließend überprüft, sodass bei Erhöhung der Werte über einen gewissen Zeitraum eine Alarmfunktion ausgeführt werden kann.

Der Arzt als Akteur hingegen ruft die Werte mithilfe eines Web-Browsers vom Server ab, welche gegebenenfalls zuvor entsprechend den Anforderungen aufbereitet wurden.

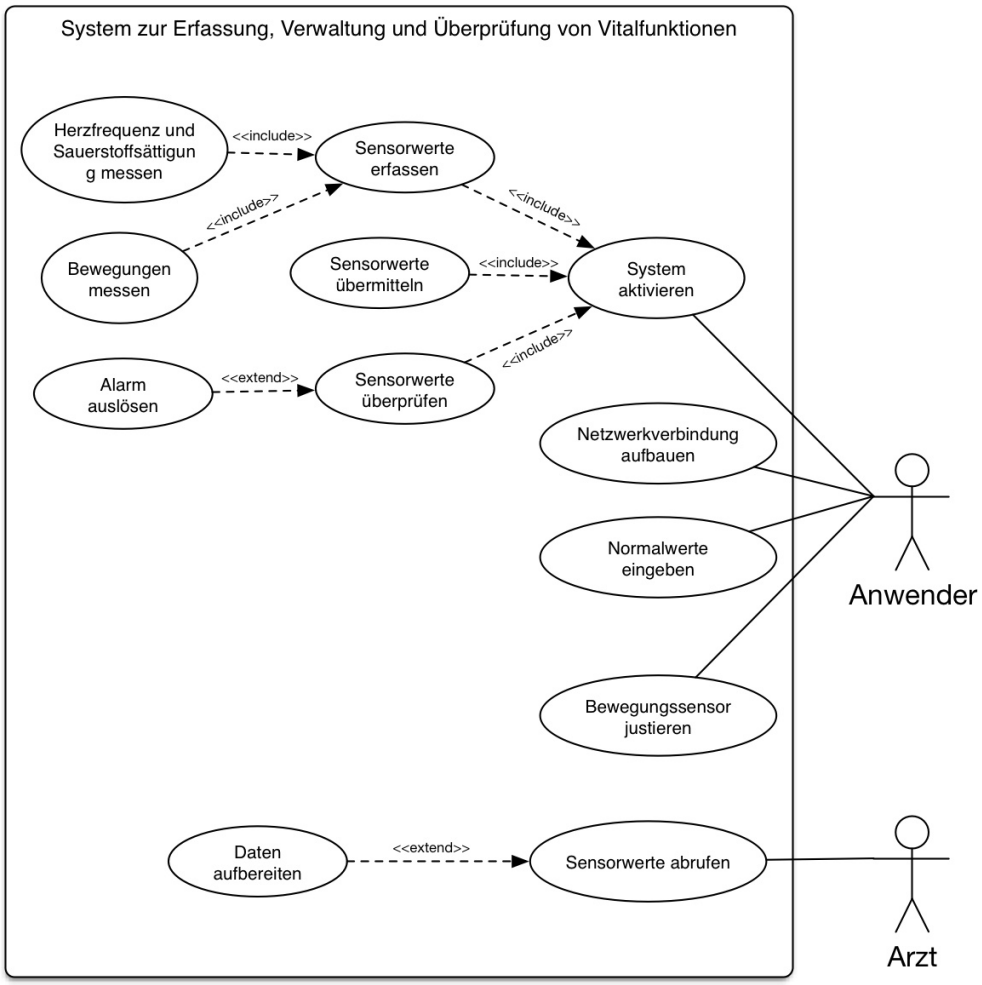


Abbildung 49: Use-Case Diagramm

3.6 Zustandsautomat

Der Zustandsautomat zeigt auf, wie das System sich bei bestimmten Ereignissen verhalten soll. Hierbei wird das System nicht für die Prototyp-Entwicklungsphase, sondern das finale, marktreife Produkt in Betracht gezogen. Dieses verfügt zum Beispiel über eine `alarm()`-Funktion, die beim Prototyp hingegen nicht vorgesehen ist. In der, an dieses Kapitel anschließenden, Konzeptionsphase wird kein weiterer Zustandsautomat für den Prototyp entwickelt, da der unten aufgezeigte Zustandsautomat die wichtigsten Funktionen abdeckt und in der Konzeption lediglich die zu implementierenden Funktionen beschrieben werden.

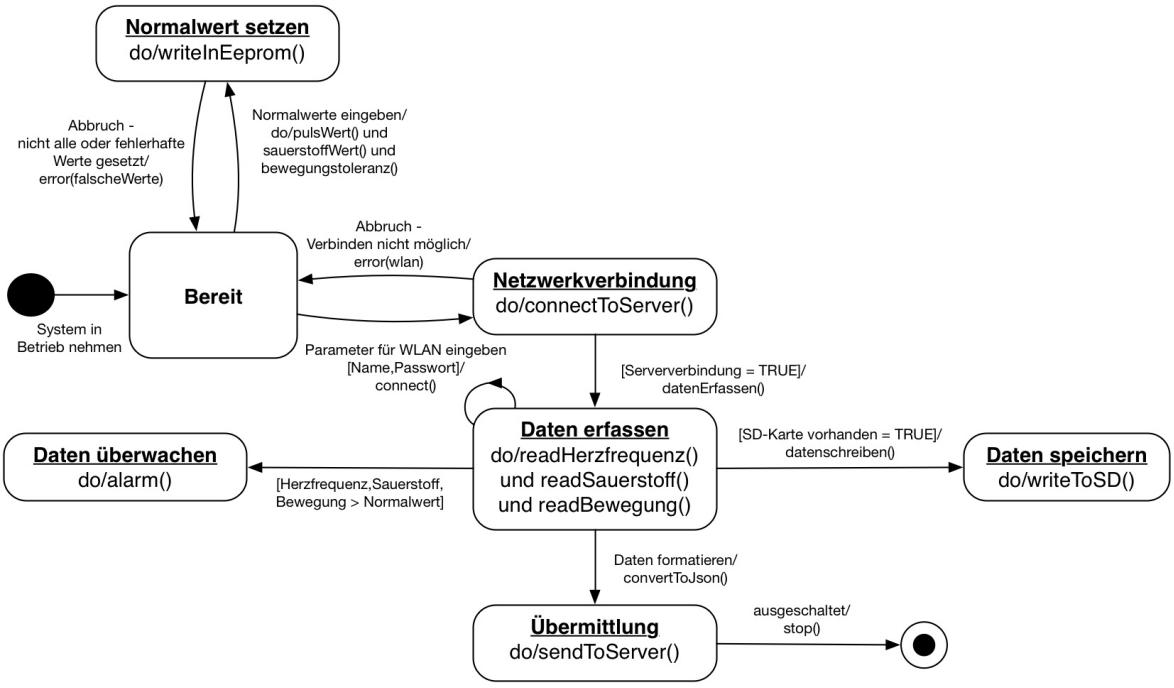


Abbildung 50: Zustandsautomat

3.7 Zusammenfassung

Mit Arduino lassen sich große Teile der Anforderungen an ein Produkt realisieren. Dabei dient Arduino allerdings als Werkzeug zur Produktentwicklung und wird als System nicht in einer medizinischen Anwendung verbaut. Durch Erweiterungs-Shields und zusätzliche Komponenten lassen sich mit einem Arduino-Board eine produktnahe funktionale Anwendung umsetzten. So auch unter dem in Projekt-Scope beschriebenen Anwendungsfall.

3.7.1 Anforderungen an den Prototyp

Folgende Anforderungen sollen in einer prototypischen Implementierung umgesetzt werden.

- Verwendung von Puls-Oxigen-Sensor und Accelerometer zur Datenerfassung
- Messung und Erfassung durch analoge und digitale Eingänge
- Kommunikation im Heimnetzwerk über eine Funkverbindung
- Verarbeitung der Daten und Vermittlung innerhalb des Heimnetzwerkes an ein IEEE 802.15.4 zu TCP-IP Gateway
- Formatierung der erfassten Daten in ein für einen PaaS-Service lesbares Format
- Nach Fertigstellung ohne Fremdeinfluss selbstständig lauffähig
- Ausreichende Speicherkapazität des Boards für den Programmcode

4 KONZEPT

Die Umsetzung des, auf der Arduino-Plattform basierenden, Prototypen im Gesundheitswesen-Bereich konzentriert sich im Wesentlichen auf den Funktionalitätsumfang. Durch Integration verschiedener Komponenten und Erweiterungen soll ein Arduino-Board um weitere Funktionen ergänzt werden, welche zum Messen von Vitalfunktionen und Bewegungsabläufen dienen.

Die Auswertung und Visualisierung der gewonnenen Daten soll über einen Servicedienstleister einfach und prototypisch implementiert werden. Ein eigenes Datenbanksystem wie auch eine eigens entwickelte Webseite ist in der Prototypen-Phase nicht relevant und somit nicht Bestandteil dieser Thesis.

Die Grundfunktionen bilden hierbei die Kommunikation über den ZigBee-Standard oder den vergleichbaren IEEE 802.15.4-Standard und decken den Informationsaustausch zwischen Sendermodul (Armband) und Empfängermodul (Gateway) ab. Die Messung der Werte soll über zwei Sensoren (Puls-Oxigen-Sensor und Accelerometer) realisiert werden. Als Plattform im Internet soll ein PaaS-Service hinzugezogen werden, über den die übermittelten Werte zeitdiskret anzeigt werden können.

4.1 Systemarchitektur und Komponenten

Die Erstellung der Systemarchitektur und die Auswahl der zu verwendenden Komponenten bildet die hardwareseitliche Entwicklung des Systems. Hierzu müssen folgende Aspekte beachtet werden.

- Ein Arduino-System muss als Produzent arbeiten und durch eine Batterie mit Strom versorgt werden können.
- Zusätzlich muss es Daten durch Sensoren erfassen und
- über ein XBee-Modul an einen Empfänger in Reichweite senden.

- Ein anderes Arduino-System fungiert als Gateway ins Internet und
- muss ebenfalls über ein XBee-Modul verfügen, um die vom Produzenten gesandten Daten empfangen zu können.
- Anschließend werden über eine WiFi- oder Ethernet-Erweiterung die Daten per HTTP-Request an eine Service-Plattform gesendet und visualisiert.
- Dieses System wird als Client bezeichnet, da es über das TCP-Protokoll mit einem Server kommuniziert.

4.1.1 Architektur

Die Systemarchitektur zeigt die einzelnen Systeme, die zu einer Kommunikation zwischen dem Arduino-Produzenten und einem Server notwendig sind.

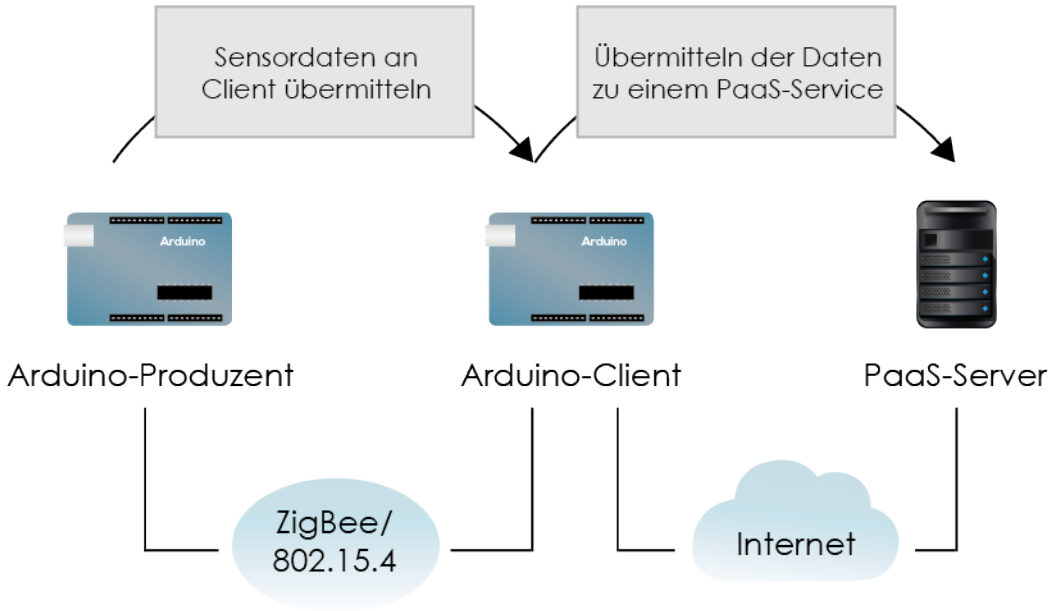


Abbildung 51: Systemarchitektur - Arduino-Anwendung im Gesundheitswesen

Der Produzent sendet über den IEEE 802.15.4-Standard, die von den Sensoren übermittelten Werte, an den Arduino-Client. Dieser ist mit dem Verbindungsaufbau und dem Erhalt der Verbindung zu einem Server beauftragt und übermittelt nach erfolgreichem Empfangen, der von dem Produzenten übertragenden Daten, diese an den Server weiter.

4.1.2 Hardware-Komponenten

Die notwendigen Komponenten werden folglich den beiden Arduino-System zugeteilt.

Arduino-Produzent

Für dieses System sind folgende Komponenten notwendig:

- **Arduino-UNO R3**

- **E-Health Sensor-Shield**
  Ein von »cooking hacks[67]« entwickeltes Shield für Arduino, um biometrische und medizinische Anwendungen für Körperfunktionsüberwachung zu realisieren.

- **Puls-Oxigen-Sensor von Contec**
  Sensor zum Messen für Puls und Sauerstoffsättigung von der Firma »Contec Medical Systems CO.«. Dieser Sensor verfügt über eine Typ BF[68]-Zulassung und eignet sich daher für den physikalischen Kontakt zum Patienten.

[67] Plattform für DIY und Internet of Things Projekte.
[68] Kategorie BF, Akronym für Body Float

- **Arduino XBee-Pro-Shield von Libelium**
  Dieses Shield ist kompatibel zum E-Health Sensor-Shield und unterstützt alle gängigen XBee-Module. Des Weiteren verfügt dieses Shield über eine Aussparung, sodass der Anschluss für den Puls-Oxigen-Sensor des darunter befindlichen Sensor-Shields zugänglich bleibt.

- **XBee S1 mit 1mW Trace(PCB)-Antenne**
  Das 2,4GHz XBee S1-Modul wird auf dem XBee-Shield angebracht und verwendet den IEEE 802.15.4-Standard. Über dieses Modul wird eine einfache Punkt-zu-Punkt Verbindung zwischen den beiden Systemen mit einer Reichweite von bis zu 100m realisiert. Dieses Modul ist nur zum seriellen Senden von Daten zuständig.

- **Sparkfun Accelerometer / 3 Axis – MMA8452Q**
  Dieser Beschleunigungssensor misst die x-,y- und z-Achse und wird über I2C mit dem System verbunden. Dadurch sollen Bewegungen beziehungsweise die Muskelzuckungen des Patienten erfasst werden.

Arduino-Client

Der Arduino-Client ist für die Verarbeitung und Vermittlung der Sensordaten zuständig und fungiert zudem als Gateway zwischen den beiden Kommunikationsstandards.

Hierfür nötige Komponenten sind folgende:

- **Arduino-UNO R3**

- **Arduino WiFi-Shield**
  Durch dieses Shield wird eine TCP-Verbindung über den Router zu einem Server hergestellt. Ein WiFi-Shield eignet sich gegenüber einem Ethernet-Shield dahingehend, dass diese Variante nicht durch ein Kabel mit dem Router verbunden werden muss und dadurch frei positionierbar ist. Durch diese Lösung sind größere Distanzen zwischen Router und Produzent möglich.

- **Sprkfun XBee-Shield**
  Dieses Erweiterungs-Shield erfüllt die dieselben Funktionen wie das Arduino XBee-Pro-Shield beim Arduino-Produzenten. Allerdings wird in diesem System keine Aussparung benötigt, weshalb dieses Shield eingesetzt werden kann. Der Leistungsumfang des Shields unterscheidet sich allerdings nicht.

- **XBee S1 mit 1mW Trace(PCB)-Antenne**
  Auch dieses Modul ist das Selbige wie beim Produzenten. Die Funktion hingegen ist das Empfangen statt Senden von Daten.

4.2 Server - Speichern und Visualisieren der Daten

Um Anwendern die gewonnenen Daten visualisiert zur Verfügung zu stellen, wird bei der Implementierung auf den PaaS-Service namens »Xively« zurückgegriffen. Mithilfe dieses Services können mit geringem Aufwand die Daten visualisiert werden. Hierzu ist lediglich ein kostenloser, aktiver Account notwendig. Die gewonnenen Daten sollen in drei

Graphen visualisiert werden. Darüberhinaus kann ein von Xively bereitgestelltes JavaScript verwendet werden, um die Daten auf einer eigenen Webseite zu visualisieren und somit auch anderen Nutzern zugänglich zu machen.

4.3 Aktivitäten der Systeme

Beide Systeme haben ihren eigenen Programmablauf und arbeiten komplett autonom und unabhängig vom jeweils anderen System. Diese werden folgend dargestellt.

4.3.1 Arduino-Produzent

Dieses System ist für das Erfassen und Übermitteln der Sensorwerte zuständig. Dahingehend müssen folgende Funktionen umgesetzt werden.

- Pulsschlag pro Minute erfassen
- Sauerstoffsättigung messen
- Bewegungen in X-,Y- und Z-Achse erfassen
- Gewonnene Daten seriell übertragen

4.3.2 Arduino-Client

Der Client ist für die Übermittlung der Daten an den Xively-Dienst zuständig. Hierzu müssen folgende Aufgaben mithilfe des Arduino-Systems erfüllt werden.

- Übertragene Daten seriell einlesen
- Einzelne Werte entsprechenden Variablen zuweisen
- Verbindung zu Xively aufbauen
- Daten in ein für die Xively-API lesbares Format bringen
- Übertragen der Daten mittels HTTP-Request

4.4 Projektumfang der Thesis

Im Rahmen dieser Thesis soll für beide Systeme eine prototypische Umsetzung erfolgen. Da es sich bei Arduino um eine Prototyping-Plattform handelt, wird wie in der Anforderungsanalyse geschildert, kein Bezug auf Ergonomie und Minimierung des Systems genommen. Diesbezüglich sollen die für Arduino erhältlichen Shields und Komponenten verwendet werden, welche im Kapitel 4.1.2 aufgeführt wurden.

Um zu schildern wie die einzelnen Komponenten miteinander interagieren, werden während der Umsetzung einzelne Tests zu den jeweiligen Szenarien durchgeführt. Hierbei soll ersichtlich werden, wie die Komponenten zu konfigurieren sind und auf Funktionalität überprüft werden können.

Der finale Prototyp soll den Anforderungen aus Kapitel 3.7.1 (Anforderungen an den Prototyp) entsprechen und Arduino im Kontext einer Anwendung im Gesundheitswesen aufzeigen.

5 IMPLEMENTIERUNG

Die Implementierung von Arduino in eine Anwendung zum Monitoring im Bereich Epilepsie, wie in der Anforderungsanalyse beschrieben, erfolgt in drei Umsetzungsphasen. In der ersten Phase wird der Arduino-Produzent umgesetzt. Hierfür sollen gemäß der testgetriebenen Entwicklung die einzelnen Module beziehungsweise Systeme getestet und für die anschließende Implementierung in das System vorbereitet werden. Dadurch kann eine Fehlfunktion im späteren Verlauf der Implementierung aufseiten der Hardwarekomponenten ausgeschlossen werden. Anschließend wird in der zweiten Phase der Xively-Account eingerichtet, sodass dieser im darauffolgenden Kapitel bei der Umsetzung des Arduino-Client implementiert werden kann.

In der dritten Phase wird der Arduino-Client entwickelt, welcher letztendlich die übermittelten Daten des Arduino-Produzenten verarbeitet und über ein IEEE 802.11b/g Netzwerk an den Xively-Server per HTTP-Request sendet. In dieser Phase der Implementierung werden beide Systeme zudem zusammengeführt und ein erster kleinerer Systemtest durchgeführt. Die vierte Phase ist nicht mehr konkreter Teil der Umsetzung, sondern fungiert als Testphase, in welcher der Prototyp bezüglich den Anforderungen in einem großen Systemtest überprüft wird. Erst durch diese Phase ist eine erfolgreiche Implementierung von Arduino als Monitoring-Anwendung festzustellen und zu belegen.

5.1 Überblick

In diesem Kapitel werden alle wichtigen Komponenten für die Entwicklung des Prototypen aufgelistet und erläutert. Des Weiteren wird die Vorgehensweise, die Architektur des Systems wie auch die einzusetzende Software erklärt.

5.1.1 Software

Entwickelt wird mit der aktuellen, stabilen Entwicklungsumgebung »Arduino 1.0.5«. Diese wird im Verlauf durch weitere Funktionen ergänzt, die durch Bibliotheken integriert werden. Zum Testen von Komponenten während der Entwicklungsphase wird auf den in der IDE integrierten Seriellen-Monitor zurückgegriffen, um Kontrollwerte ausgeben zu lassen. Darüberhinaus wird die Funktionalität des Accelerometer über eine Processing-Anwendung dargestellt, wodurch die erfassten Werte grafisch visualisiert werden. Hierzu wird »Processing 2.0.3« verwendet, da Version 2.1.X über einen Bug in der Funktion `readStringUntil()` verfügt und somit das Auslesen der eintreffenden Zeichenkette erschwert. Zum Konfigurieren der XBee-Module stehen mehrere Softwarelösungen zur Verfügung. Für gewöhnlich wird hierzu die vom Hersteller der XBee-Module bereitgestellte Software namens »X-CTU« verwendet. Diese ist allerdings nur zu einem Windows-Betriebssystem kompatibel. Da der bei der Entwicklung verwendete Computer mit OS X Mavericks arbeitet, wird auf eine kostenlose Anwendung namens »CoolTerm« zurückgegriffen, welche die Kommunikation zwischen Terminal und seriellem Port verwaltet. Diese Anwendung ist für Mac, Win und Linux erhältlich.

5.1.2 Architektur und Module

Die einzelnen Komponenten und Module zur Entwicklung des Prototyps wurden im Kapitel 4.1.2 (Hardwarekomponenten) vorgestellt und werden im Laufe der Implementierung genauer beschrieben. Folgende Grafik zeigt die Kommunikationsarchitektur der einzelnen Systeme und Komponenten. Hierbei handelt es sich bei dem oberen System um den Arduino-Produzent, welcher die Sensoren ausliest und über ein XBee-Modul diese an den Client (unteres System) übermittelt. Dieses System ist mit der Verarbeitung des eintreffenden Textstrings beauftragt und stellt über ein WiFi-Shield eine Verbindung zum Visualisierungs-Service Xively her.

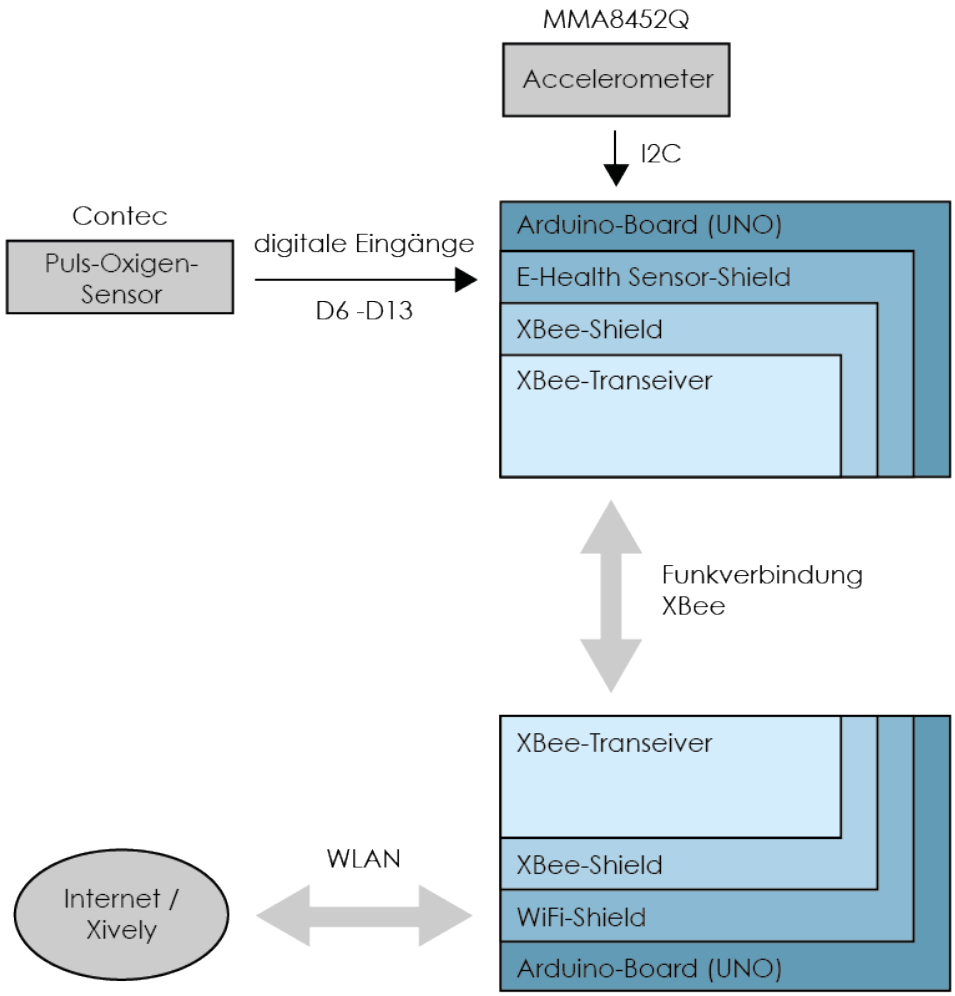


Abbildung 52: Kommunikationsarchitektur des Prototypen

Module Produzent

- Arduino-UNO R3
- E-Health Sensor-Shield
- XBee-Shield Arduino/Libelium
- XBee S1 (Transeiver)
- Accelerometer MMA8452Q von Sparkfun
- Puls-Oxigen-Sensor von Contec

Module Client

- Arduino-UNO R3
- WiFi-Shield R3 von Arduino
- XBee-Shield von Sparkfun
- XBee S1 (Transeiver)

Für Konfiguration relevante Module und Komponenten

- XBee Explorer USB von Sparkfun:
  Eine einfache USB-zu-Serial Basiseinheit zur Konfiguration jeglicher XBee-Module.
- USB Typ-A auf Mini-B Kabel:
  Wird benötigt um den XBee Explorer an den Computer anzuschließen.
- USB Typ-A auf Typ-B Kabel:
  Zum Verbinden der Arduino-Boards mit dem Computer.
  Andere Arduino-Modelle arbeiten mittlerweile mit Typ-A auf Micro-B USB-Kabel.
- Arduino Stackable Header-Kit für R3
- Break-Away Headers - Long
- 9V zu Barrel-Jack Adapter und 9V Blockbatterie
- 9V Netzteil Barrel-Jack
- Lötdraht und Lötkolben

5.1.3 Testgetriebene Entwicklung und Debugging

Dieser Begriff wird für gewöhnlich in der Softwareentwicklung verwendet und beschreibt eine Entwicklungsmethode. Während nach der klassischen Methode beziehungsweise Softwareentwicklung, Tests dahingehend verwendet werden, um die Funktionalität fertiger Komponenten sicherzustellen, werden bei der testgetriebenen Entwicklung die Tests noch vor dem eigentlichen Produktivcode geschrieben. Hierbei wird in zwei Kategorien unterschieden und getestet.

- **Unit-Test**
  Auch Modul- oder Komponententest genannt, befasst sich mit Tests an funktionalen Einzelteilen (Module).

- **Systemtest**
  Hierbei wird das gesamte System gegen die gesamten Anforderungen getestet. Die Testumgebung soll dabei der Produktivumgebung des Kunden ähneln beziehungsweise soll diese simuliert werden.

Der Systemtest wird in dieser Thesis als abschließender Testlauf durchgeführt und muss als Ergebnis, die durch die Anforderungsanalyse ermittelten Anforderungen an den Prototyp erfüllen. [112]

Die Implementierung des Prototypen wird auf Basis der testgetriebenen Entwicklung umgesetzt, sodass durch die Tests die Funktionalität der Module dokumentiert wird. Allerdings werden die Module und Komponenten auf die für sie bestimmte Funktion überprüft, sodass bei der Implementierung volle Funktionalität bei den einzelnen Modulen

gewährleistet werden kann. In diesem Zusammenhang wird ebenfalls die technische Funktionalität der elektrischen Schaltungen überprüft.

Zum Testen und Debuggen werden folgende Methoden verwendet.

- Daten, Zustände oder Variablenwerte können mit Hilfe von `Serial.print()` an den Seriellen-Monitor übermittelt werden. Diese Methode eignet sich vor allem zum Testen von Aktionen und Funktionen wie auch der Darstellung von Signalen. Zudem wird auf Processing zur Visualisierung von Daten im zeitlichen Verlauf zurückgegriffen.
- Fehler im Programmcode wie ein fehlendes Semikolon oder vergessene Deklaration werden durch den Editor beim Kompilieren des Programmcodes erkannt und in einem Infofenster ausgegeben.

Liegt es allerdings nicht an der Syntax des Codes, so können Fehler auch im Programmablauf entstehen und zum Beispiel Funktionen nicht aufgerufen und somit ausgeführt werden. Um diese Fehler zu erkennen, wird der Serielle-Monitor verwendet, der Daten bei der Fehlersuche im Code sichtbar macht.

5.1.4 Aufbau

Für jedes Modul und jede Komponente wird ein eigener Testlauf bezüglich der Funktionalität für die anschließende Implementierung in das System durchgeführt. Gemäß der
Abbildung 52 werden die Systeme für den Produzenten und Client separat und von innen heraus entwickelt. Das bedeutet, dass zuerst die Kommunikation zwischen den XBee-Modulen realisiert werden soll und anschließend das Erfassen der Sensordaten wie auch deren Übermittlung. Die Basis für die Konfiguration und die Tests bilden hierbei die beiden Arduino-Boards mit dem ATmega328 Mikrocontroller. Um die erfolgreiche Implementierung in das System ansatzweise vorzubereiten, müssen die eingesetzten Module gemäß dem späteren Anwendungsgebiet und Funktionalitätsumfang entwickelt und getestet werden.

5.2 Phase 1: Umsetzung Arduino-Produzent

In diesem Kapitel werden die einzelnen Module und Komponenten für den späteren Einsatz konfiguriert und getestet. Hierbei werden die Verfahren erläutert und die Ergebnisse beschrieben. Zuerst werden die für die Funkverbindung relevanten XBee-Module eingerichtet, welche für die Kommunikation zwischen den beiden Systemen nötig sind. Anschließend wird der Arduino-Produzent entwickelt und getestet. In den darauffolgenden Kapiteln wird der Xively-Service konfiguriert und der Arduino-Produzent umgesetzt.

5.2.1 Konfiguration der XBee-Module

Da es sich bei der Kommunikation um eine Punkt-zu-Punkt Verbindung handelt, ist kein vermarschtes Netzwerk nötig und es kann gemäß des IEEE 802.15.4-Standards entwickelt werden.

Benötigte Hardware

- Zwei XBee S1-Module
- XBee Explorer USB
- USB-Kabel Typ-A auf Mini-B

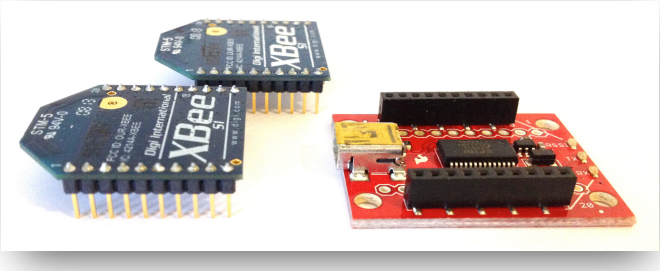


Abbildung 53: XBee Explorer mit zwei XBee S1 Modulen

Benötigte Software

- CoolTerm

Umsetzung

Die beiden Module müssen separat voneinander eingerichtet werden. Hierzu müssen diese auf dem XBee Explorer aufgesteckt und dieser im Anschluss durch das USB-Kabel mit dem Computer verbunden werden. Anschließend wird CoolTerm ausgeführt. Mit dieser Software kann das XBee-Modul über den seriellen Port beschrieben werden. Um eine Punkt-zu-Punkt Verbindung für die beiden Module einzurichten, genügen lediglich wenige Befehle aus dem »XBee-Datasheet«, erhältlich auf der Webseite des Herstellers »Digi«.

Für die Konfiguration wichtige AT[69]-Kommandos:

- **WR:** Schreibe Parameterwerte auf Speicher des Moduls.
- **RE:** Parameter auf Werkseinstellung zurücksetzten.
- **ID:** Setzten und Lesen der PAN[70]-ID.
- **BD:** Setzten und Lesen der Baudrate der seriellen Schnittstelle.
- **+++:** Dieser Befehl ist nicht Teil der AT-Befehle sondern führt einen Wechsel in den XBee-Kommandomodus durch.

[69] Der AT-Befehlssatz ist quasi ein Standard zum Konfigurieren und Parametrieren von Modems.
[70] Akronym für Personal Area Network

Das Sender- wie auch Empfänger-Modul wird gleichermaßen konfiguriert. Hierzu wird unter CoolTerm die Funktion »Send String« ausgewählt, wodurch dem XBee-Modul ein ASCII-String gesendet werden kann. Über folgende Befehlseingaben wird das Modul konfiguriert.

```
+++              //Wechseln in den XBee-Kommandomode
ATRE             //Modul auf Werkseinstellung zurücksetzen
ATID             //Aktuelle Netzwerk-ID abfragen
ATID 2000        //Neue ID = 2000 festlegen
ATID             //ID überprüfen
ATBD             //Baudrate abfragen
ATWR             //Die Parameterwerte auf Speicher schreiben
```

Quellcode 34: XBee-Modul Konfiguration

Für jede Eingabe liefert das XBee-Modul eine Antwort zurück. Alle AT-Befehle müssen hierbei mit einem Carriage-Return enden. Entsprechend den oben eingegeben Befehlen sehen die vom XBee zurückgelieferten Antworten wie folgt aus.

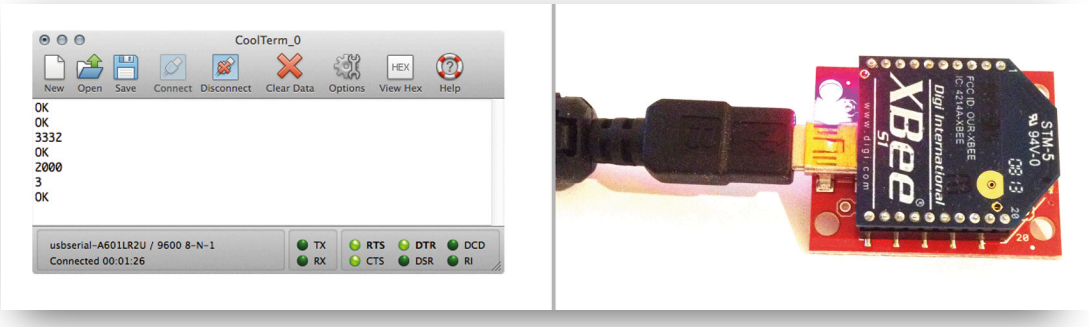


Abbildung 54: CoolTerm Ausgabe bei XBee-Konfiguration

Der Default-Wert für die Netzwerk-ID liegt bei jedem XBee-Modul ab Werk auf »3332«. Dieser wird in »2000« geändert und anschließend die Baudrate abgefragt. Diese muss mit der im späteren Arduino-Sketch übereinstimmen. Der Wert »3« steht für eine Baudrate von 9600 und ist aus dem Datasheet von XBee zu entnehmen. Dieser Wert wird vorerst beibehalten. Sollte dieser verändert werden, so muss der AT-Befehl **`ATBD X`** (X steht für entsprechende Nummer aus Datasheet) ausgeführt werden. Durch **`ATWR`** wird letztendlich die Parametrierung auf dem Speicher des XBee-Moduls durchgeführt und ist dadurch auch nach Trennen der Stromversorgung persistent.

Testen der Module

Nachdem beide Module mit der Netzwerk-ID von 2000 und einer Baudrate von 9600 konfiguriert wurden, können diese getestet werden. Hierzu muss ein Arduino-Board in Verbindung mit einem der beiden XBee-Module eine serielle Ausgabe schreiben, sodass das Empfängermodul, angeschlossen über den XBee Explorer am Computer, die

übermittelten Werte in einer Terminal-Anwendung ausgeben kann. Hierbei wird ebenfalls CoolTerm verwendet.

Auf den Arduino-UNO wird folgender Programmcode übertragen:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print('H');
  delay(1000);
  Serial.print('L');
  delay(1000);
}
```

Quellcode 35: Arduino XBee-Sender Programmcode

Dieser sorgt dafür, dass in einer Schleife erst ein »H« ausgegeben, dann eine Sekunde gewartet und anschließend ein »L« ausgegeben wird. Diese Abfolge wiederholt sich, solange das Sender-Modul mit Strom versorgt wird.

In der CoolTerm-Anwendung ist demnach Folgendes zu beobachten.

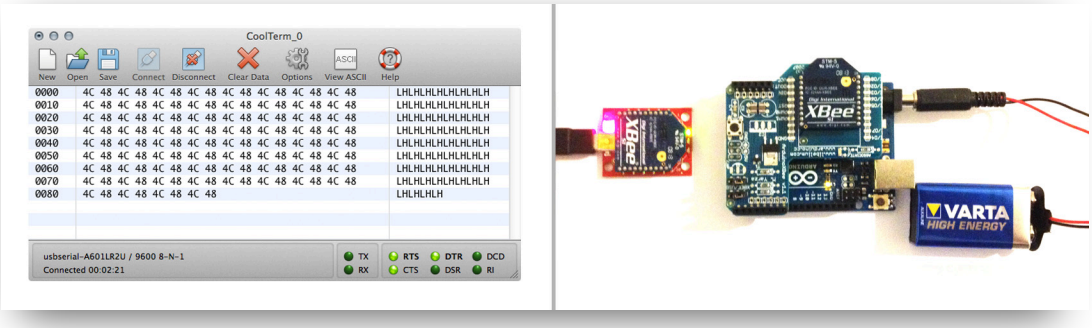


Abbildung 55: CoolTerm XBee überprüfen

In der rechten Spalte des CoolTerm-Terminals sind die sekündlich erscheinenden ASCII-Zeichen zu erkennen, welche kontinuierlich vom Arduino-Sender (rechtes System in der Abbildung) übermittelt werden. Diese stimmen mit den vom Sender übermittelten ASCII-Zeichen überein.

Da das Sender- und das Empfänger-Modul demnach richtig arbeiten, ist der Test bestanden und die Funktionalität und richtige Konfiguration der Module nachgewiesen.

5.2.2 Testen des Accelerometer

In dieser Phase der Umsetzung soll der Accelerometer beziehungsweise Beschleunigungssensor auf Funktionalität überprüft und anschließend gemäß den für den Prototyp verlangten Anforderungen in einem Arduino-Sketch programmiert werden.

Dieser soll lediglich Bewegungen in x-, y- und z-Achse erfassen und ab einer gewissen Beschleunigung (später das Zucken der Gliedmaßen) die Meldung ausgeben, dass eine Bewegung vorliegt.

Durch einen Parameter sollen hierbei langsame, im Schlaf natürliche Bewegungsabläufe erkannt und nicht als Zuckung, hervorgerufen durch einen epileptischen Anfall, ausgeben wird. Hierbei wird nicht die relative Beschleunigung gemessen, sondern lediglich die Beschleunigung in Bezug zur Erdbeschleunigung »G (9,81 m/s^2)«. Der Parameter legt das Vielfache der Erdbeschleunigung fest, ab welcher eine Bewegung erkannt werden soll. Vielfaches bedeutet in diesem Fall zum Beispiel die doppelte Erdbeschleunigung »2 G«. Je höher dieser Wert eingestellt wird, desto schnellere Bewegungen mit Richtungswechsel beziehungsweise Beschleunigungen sind nötig, damit die Funktion `TRUE` zurückgibt. Nach diesem Verfahren können normale Bewegungsabläufe sehr gut aussortiert werden.

Benötigte Hardware

- MMA8452 Accelerometer
- Arduino-UNO
- USB-Kabel Typ-A auf Typ-B
- Jumper-Kabel
- Steckboard

Benötigte Software

- Arduino 1.0.5
- Processing 2.0.3 (Wichtig, nicht 2.1.X)

Benötigte Bibliothek für Arduino

- **MMA8453_n0m1 Library**
  Dies ist eine von Noah Shibley entwickelte Bibliothek für die Verwendung des MMA8453-Beschleunigugnssensor und beinhaltet Funktionen zum Abrufen und Verarbeiten der gewonnenen Daten. Die Bibliothek steht unter folgendem Link zum Download bereit: *https://github.com/n0m1/MMA8453_n0m1*
- **Wire**
  Wire ist eine in der Arduino-Entwicklungsumgebung integrierte Bibliothek für TWI beziehungsweise I2C Kommunikation.

Umsetzung

Der Accelerometer wird von Werk aus ohne Steckleiste ausgeliefert. Daher müssen zuallererst sogenannte »Break-Away Headers« angelötet werden, damit der Sensor wie in folgender Abbildung auf das Steckboard angebracht werden kann. Die Kabel lassen sich auch direkt an den Steckplätzen anlöten, allerdings ist in der Entwicklungsphase davon abzuraten, damit das System weiterhin flexibel bleibt.

Anschließend muss der Sensor an das Board angeschlossen werden.
Da es sich um einen TWI oder I2C Sensor handelt, muss SCL des Sensors an Pin A5 und SDA an Pin A4 des Arduino-Boards angeschlossen werden. Zudem wird der Sensor mit 3,3 Volt versorgt. Um später mithilfe der Bibliothek von Shibley die Funktionen zur

Erkennung von Bewegungen ausführen zu können, wird zudem ein Interrupt-Port[71] benötigt. Daher muss der Sensoranschluss »I1« (Interrupt 1) mit Pin D2 des Boards verbunden werden. Dies wird durch die Spezifikationen bezüglich der Verwendung der Bibliothek vorgeschrieben.

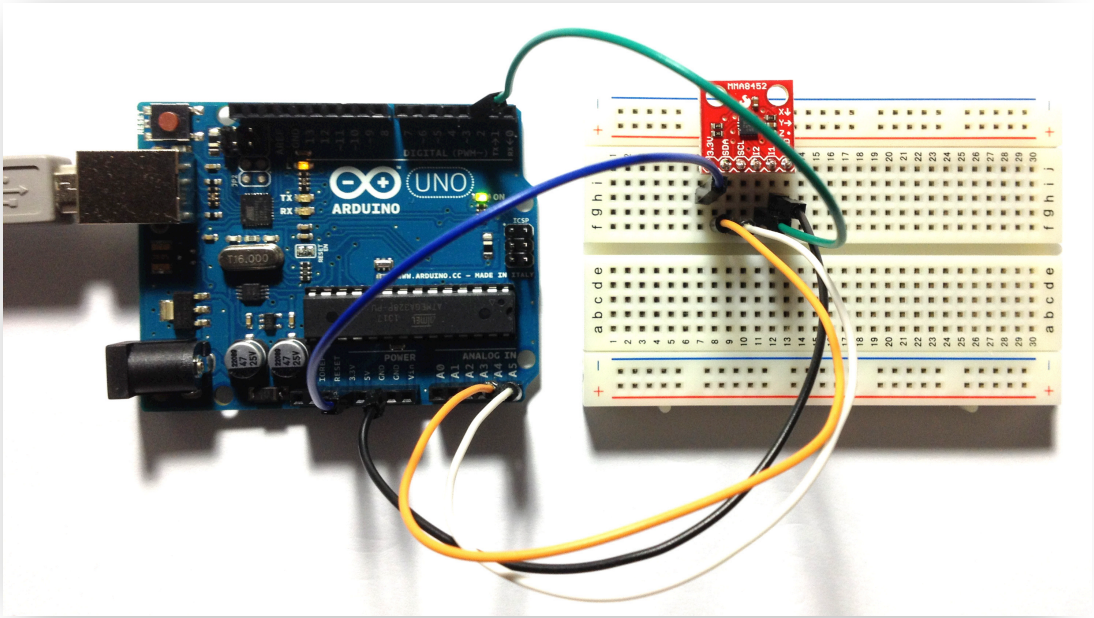


Abbildung 56: Arduino mit Accelerometer

Nachdem der Sensor gemäß Abbildung 56 am Arduino-UNO angeschlossen und dieser mit dem Computer verbunden ist, kann ein erster Arduino-Sketch zum Testen des Sensors geschrieben werden. Über den seriellen Port kommuniziert das Arduino-Board mit dem Computer. Daher werden die Ausgaben mittels `Serial.print()` an den Computer übertragen. Diese Kommunikationsvariante wird auch später für die Übermittlung der Daten über die XBee-Module verwendet.

Es wird direkt mit der oben aufgeführten Bibliothek gearbeitet. Daher muss diese, wie in Kapitel 2.6 beschrieben in die Arduino-Entwicklungsumgebung implementiert werden. Nach einem Neustart von Arduino stehen vier Beispiel-Sketche und die Verwendung der Bibliothek zur Verfügung. Auch wenn diese ursprünglich für den Accelerometer MMA8453 geschrieben wurde, ist diese für die uneingeschränkte Verwendung mit dem MMA8452 ebenfalls einsetzbar. Um zu überprüfen, ob alle Achsen des Accelerometers korrekt angeschlossen und gemessen werden können, wird vorerst ein Sketch für die Ausgabe via Serieller-Monitor geschrieben. Hierzu wird der Beispiel-Sketch »MMA8453_n0m1_dataMode« geöffnet. In diesem Beispiel ist die Wire-Bibliothek für die I2C-Verbindung wie auch die entsprechende MMA8453-Bibltiohek inkludiert.

[71] Durch Interrupts wird das Hauptprogramm gestoppt und ein anderes Programm ausgeführt wenn ein festgelegtes Ereignis an einem bestimmten Pin beziehungsweise Port auftritt. Durch Interrupts wird die Leistung des Mikrocontrollers weniger beansprucht wenn zyklische Abfragen stattfinden. Somit wird ein Sensor nicht kontinuierlich abgefragt, sondern erst wenn das Ereignis eintritt, ansonsten geht das Programm dem normalen Programmablauf nach. [2]

```
#include <Wire.h>
#include <MMA8453_n0m1.h>

MMA8453_n0m1 accel;

void setup() {
  Serial.begin(9600);
  accel.setI2CAddr(0x1D);
  accel.dataMode(true, 2);
}

void loop() {
  accel.update();

  Serial.print("X: ");
  Serial.print(accel.x());
  Serial.print(" Y: ");
  Serial.print(accel.y());
  Serial.print(" Z: ");
  Serial.println(accel.z());
}
```

Quellcode 36: Beispiel-Sketch dataMode von Noah Shibley

In diesem Sketch wird zuerst ein Objekt vom Typ MMA8453_n0m1 namens »accel« erzeugt. Anschließend wird die Baudrate für die serielle Kommunikation auf 9600 eingestellt und über `accel.setI2CAddr(0x1D)` die Geräteadresse des Sensors festgelegt. Diese kann frei gewählt werden, da allerdings nur ein Sensor über I2C angeschlossen wird, kann der Default-Wert beibehalten werden.

Über `accel.dataMode(true,2)` wird über den booleschen Wert `true` die Auflösung der Sensorwerte auf »High« (10 Bit) gesetzt. Der zweite Parameter, in diesem Fall `2`, legt den Kräftebereich und somit Sensibilität des Sensors auf die zweifache Erdbeschleunigung fest.

Mit `accel.update()` werden zu Beginn der Schleife die Sensorwerte aktualisiert und die Interrupts gelöscht. Anschließend werden über `Serial.print()` die Sensorwerte nacheinander ausgegeben. Hierzu werden über die Funktionen `x()`, `y()` und `z()` die jeweiligen Sensorwerte als Integer-Wert zurückgegeben. Nachdem der Code kompiliert und auf das Arduino-Board übertragen wurde, kann auf dem Seriellen-Monitor Folgendes betrachtet werden.

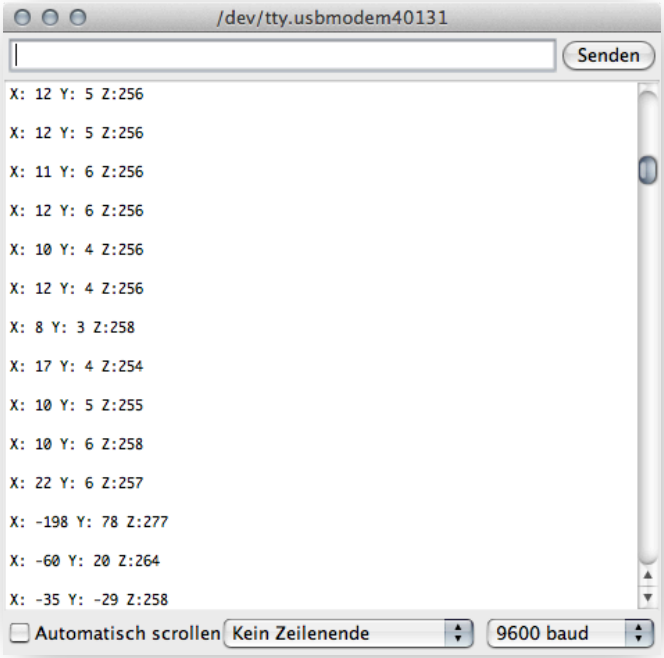


Abbildung 57: Ausgabe Serieller-Monitor, Accelerometer

Durch Bewegen des Sensors um die jeweiligen Achsen können die Veränderungen an den Werten gesichtet und somit die Funktionalität des Sensors geprüft werden. Ab der drittletzten Zeile in Abbildung 57 sind Veränderungen besonders auf der x- und y-Achse zu erkennen. Diese wurden durch Bewegen des Sensors auf dem Tisch erzeugt. Demnach scheint der Sensor korrekt zu funktionieren und angeschlossen zu sein.

Testen mit Processing

Aufgrund der eingeschränkten Darstellung auf dem Seriellen-Monitor, soll die Funktionalität des Sensors noch mit Processing visualisiert werden. Durch das Darstellen der Werte im zeitlichen Verlauf mittels eines Graphen wie auch eines Quaders, welcher die Bewegungen des Accelerometers um die jeweilige Achse simuliert, soll der Sensor überprüft werden können.

Die Ausgabe ist dieselbe wie im oberen Sketch, allerdings werden nicht mehr die Namen beziehungsweise Bezeichner der Achsen ausgegeben, sondern der X- und Y-Wert mittels einer Pipe »|« getrennt und die Z-Achse zum Y-Wert mit einem Doppelpunkt »:«. Diese können mit dem Processing-Code von Christopher Hazlett (*http://www.robotishappy.com/2010/08/part-1-visualizing-3-axis-accelerometer-readings-in-processing*) direkt eingelesen und individuell in Variablen gespeichert werden, da diese als ASCII-String am Computer eintreffen und voneinander getrennt werden müssen. Auf den Processing-Code wird nicht weiter eingegangen, da dies nicht Teil dieser Thesis ist und lediglich zum Darstellen und Überprüfen der Werte des Accelerometers dient.

```
#include <Wire.h>
#include <MMA8453_n0m1.h>

MMA8453_n0m1 accel;

void setup() {
  Serial.begin(9600);
  accel.setI2CAddr(0x1D);
  accel.dataMode(true, 2);
}

void loop() {
  accel.update();

  Serial.print(accel.x());
  Serial.print("|");
  Serial.print(accel.y());
  Serial.print(":");
  Serial.print(accel.z());
}
```

Quellcode 37: Beispiel-Sketch Arduino für Processing - Accelerometer

Der Processing-Code überwacht sozusagen den seriellen Port, welcher im Programmcode angegeben werden muss. Dieser liest die eintreffenden Zeichen ein, trennt diese voneinander und speichert diese spezifisch der Achsen ab. Somit ist eine Darstellung im zeitlichen Verlauf möglich. Wird die Processing-Anwendung gestartet, so bewegt sich mit dem Sensor auch der Quader in dem Fenster gemäß der Rotation des Sensors. Auf der linken Seite in Abbildung 58 befindet sich der Graph, welcher die Werte im zeitlichen Verlauf pro Achse darstellt. Durch diesen Test ist die Funktionalität des Sensors wesentlich deutlicher darzustellen als zuvor mit dem Seriellen-Monitor. Das Resultat sieht demnach wie folgt aus.

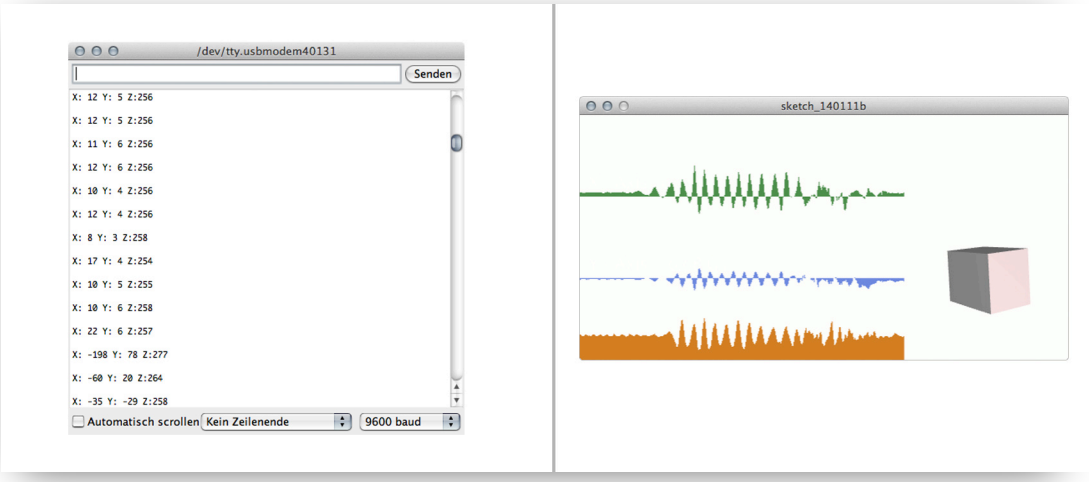


Abbildung 58: Accelerometer Processing Test

Durch diesen Versuch beziehungsweise Test wurden alle drei Achsen auf deren Funktionalität überprüft und haben den Test erfolgreich bestanden.

Umsetzung für Prototyp

Für den Prototyp ist ausschließlich die Erkennung von Muskelzuckungen relevant und es ist daher nicht nötig, die Werte der drei Achsen zu übermitteln. Lediglich ob eine Bewegung mit einer gewissen Intensität stattfand. Die Arduino-Bibliothek von Noah Shibley bietet auch für diesen Anwendungsfall eine geeignete Funktion an, welche über einen zusätzlichen Parameter für die Intensität verfügt. Die Funktion `shakeMode()` führt den transienten Erkennungsmodus aus, über den die Intensität, die Achsen wie auch der Interrupt-Pin eingestellt werden können.

Folgende Parameter stehen hierbei zur Verfügung.

- (int) Intensität [0-127]
- (boolean) aktiviere x-Achse
- (boolean) aktiviere y-Achse
- (boolean) aktiviere z-Achse
- (boolean) aktiviere Sensor Pin INT2 (false = Pin INT1)
- (int) Arduino-Board Interrupt-Pin

Die Funktion `shake()` liefert `TRUE` zurück, wenn eine Bewegung, entsprechend dem Hochpassfilter für die Bewegung (Parameter Intensität) durchgeführt wurde.

Da für den Prototyp kontinuierlich ein Wert an den Server übermittelt werden muss, wird in Ruhezustand der Wert »0« und wenn die `shake()`-Funktion `TRUE` zurückgibt der Wert »1« geschrieben. Xively benötigt Zahlen, damit Graphen gezeichnet werden können. Daher darf nicht mit Zeichen oder Ausdrücken gearbeitet werden. Der Programmcode muss demnach wie folgt aussehen.

```
#include <Wire.h>
#include <MMA8453_n0m1.h>

MMA8453_n0m1 accel;

void setup() {
  Serial.begin(9600);

  accel.setI2CAddr(0x1D);
  accel.shakeMode(4,true,true,true,false,2);
}

void loop() {
  accel.update();

  if(accel.shake()) {
    Serial.print("1");
  }
  else
    Serial.print("0");
}
```

Quellcode 38: Accelerometer Arduino-Programmcode für Prototyp

Die Funktion `accel.shakeMode(4,true,true,true,false,2)` legt fest, dass bei einer Beschleunigung von 4*0,063 g[72], somit 0,252 g eine Bewegung als Muskelzuckung erkannt werden soll. Dies entspricht wiederum zirka 2,5 m/s^{-2} (0,252*9,81 m/s^{-2}). Dem Anwender steht hierbei ein Wertebereich von 0 (0 g) bis 127 (8 g) zur Verfügung, wodurch die Sensibilität wesentlich feiner justiert werden kann als über natürliche Zahlen für einen vielfachen Wert von »G«. Die folgenden drei booleschen Werte geben an, dass alle drei Achsen gemessen werden sollen. Anschließend wird noch der Interrupt-Pin des Sensors auf INT1 gestellt und am Arduino-Board der Pin D2 als Interrupt-Pin ausgewählt, da das XBee-Shield über Pin D0 und D1 kommuniziert. Somit ist der Programmcode für den Accelerometer geschrieben und es kann der nächste Sensor getestet werden.

5.2.3 Testen des Puls-Oxigen-Sensors

Als Nächster und Letzter für den Prototyp relevanten Sensor wird der Puls-Oxigen-Sensor von Contec getestet und beispielhaft implementiert. Dieser erfasst über verschiedene Wellenlängen von Licht die Sauerstoffsättigung wie auch den Pulsschlag. Diese beiden Werte sollen für den Prototyp als Integer-Werte bereitgestellt werden, sodass diese an den Server übertragen werden können.

Benötigte Hardware

- Puls-Oxigen-Sensor von Contec
- E-Health Sensor-Shield V2.0 von cooking hacks
- Arduino-UNO R3
- USB-Kabel Typ-A auf Typ-B

Benötigte Software

- Arduino 1.0.5

Benötigte Bibliothek für Arduino

- **eHealth- und PinChangeInt-Library**
  Beide Bibltioheken stehen auf der Webseite des Herstellers des Sensor-Shields zum Download bereit: *http://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical*.

Umsetzung

Das Sensor-Shield wird direkt auf dem Arduino-Board angebracht und verbindet die benötigten Pins miteinander. Mit Strom wird das Shield wie auch der Sensor über den ICSP-Anschluss des Arduinos versorgt. Das Shield bietet eine Vielzahl an Anschlussmöglichkeiten, da es über sieben Sensoren vom Hersteller unterstützt wie auch weitere Anschlüsse durch Durchschleusen der Pins vom Arduino-Board bereitstellt. Für die Entwicklung des Prototypen eignet sich dieses Shield aufgrund der spezifischen Entwicklung für Arduino und den dadurch bereitgestellten Bibliotheken und Funktionen zum Ansprechen der Sensoren. Für marktreife Produkte hingegen kann auf das Shield

[72] Formel zur Berechnung der Werte von der Quelle der Arduino-Bibliothek [113]

verzichtet und die Sensoren direkt am Mikrocontroller angeschlossen werden. Das auf der Webseite des Herstellers erhältliche Datenblatt gibt darüber hinaus Einblicke auf die Pinbelegung und Stromversorgung der einzelnen Sensoren, wodurch diese auch direkt am Arduino-Board angeschlossen werden könnten (siehe Anhang J: Pinbelegung Arduino-UNO (Produzent)). Der Puls-Oxigen-Sensor wird über einen 10-Pin-Anschluss auf dem Shield angebracht.

Abbildung 59: Arduino-Uno, Sensor-Shield und Puls-Oxigen-Sensor

Beide Bibliotheken müssen ebenfalls in das Verzeichnis für Arduino-Bibliotheken abgelegt werden, damit diese im Programmcode verwendet werden können. Dadurch stehen folgende Funktionen für die Programmierung zur Verfügung.

- **initPulsioximeter()**
  Initialisiert den Puls-Oxigen-Sensor.
- **readPulsioximeter()**
  Liest die Werte des Puls-Oxigen-Sensors.
- **getBMP()**
  Gibt die Herzfrequenz beziehungsweise den Puls in Schlägen pro Minute zurück.
- **getOxygenSaturation()**
  Gibt die Sauerstoffsättigung im Blut in Prozent zurück.

Bei der Umsetzung des Programmcodes genügt lediglich eine Ausgabe über den Seriellen-Monitor, da das Wertepaar resultierend aus Puls und Sauerstoffsättigung später seriell über das XBee-Modul an den Client übergeben werden soll.

Auch dieser Sensor verwendet eine Interrupt-Funktion, um den Programmablauf zu unterbrechen und die Sensorwerte zu aktualisieren. Hierfür wird zuerst die PinChangeInt-Bibliothek eingebunden und im Setup anschließend die entsprechende Funktion ausgeführt.

Der Pin D6 des Arduino wird, durch `attachInterrupt(6,readPulsioximeter,RISING)`, als Interrupt-Pin festgelegt, an dem der Sensor die Unterbrechung bei Steigen des anliegenden Wertes herbeiführt und in die Funktion `readPulsioximeter()` springt. In dieser wird nur jede fünfzigste Messung in die zugehörigen, privaten Variablen geschrieben und sorgt somit für eine geringere Rechenarbeit und Zeit pro Programmdurchlauf. Demnach wird die Funktion `readPulsioximeter()` benötigt, welche durch eine Bedingung und einen Zähler dafür sorgt, dass nur bei jedem fünfzigsten Programmdurchlauf die Funktion `eHealth.readPulsioximeter()` ausgeführt wird und eine Messung und Speicherung durch den Sensor stattfindet.

Test des Puls-Oxigen-Sensors

Für den Test wird ein Arduino-Sketch gemäß den zuvor aufgeführten Funktionen erstellt und die vom Sensor gemessenen Werte am Seriellen-Monitor ausgegeben. Dadurch lässt sich die Funktionalität des Sensors überprüfen. Durch Messungen eines zweiten, medizinischen Messgerätes würden sich die Werte zudem überprüfen lassen, allerdings ist bekannt, dass Werte bei der Sauerstoffsättigung zwischen 96 und 98 Prozent als normal gelten und starke Abweichungen somit auf ein Fehlverhalten des Sensors hinweisen würden. Dies ist ebenfalls bei der Messung des Puls der Fall, bei der in Kategorien unterschieden werden muss.

- **Kinder unter 1 Jahr:** 100-160 bpm
- **Kinder zwischen 1 und 10 Jahren:** 70-120 bpm
- **Personen über 10 Jahre:** 60-100 bpm
- **Trainierte Personen/Athleten:** 40-60 bpm

Darüber hinaus können bei der Pulsoximetrie durch Umweltfaktoren oder fehlerhafte Anbringung des Sensors Fehlfunktionen vorliegen. [114]

Die oben genannten Referenzwerte lassen sich daher zur Kontrolle der Testergebnisse heranziehen.

```
#include <PinChangeInt.h>
#include <eHealth.h>

int zaehler = 0;

void setup() {
  Serial.begin(9600);
  eHealth.initPulsioximeter();
  PCintPort::attachInterrupt(6,readPulsioximeter,RISING);
}

void loop() {
  Serial.print("Puls : ");
  Serial.print(eHealth.getBPM());
  Serial.print("    Sauerstoff : ");
  Serial.print(eHealth.getOxygenSaturation());
  Serial.println();
  Serial.println("==============================");
  delay(100);
}

void readPulsioximeter() {
  zaehler ++;
  if (zaehler == 50) {
    eHealth.readPulsioximeter();
    zaehler = 0;
  }
}
```

Quellcode 39: Beispiel-Sketch Puls-Oxigen-Sensor

Durch Ausführen dieses Programmcodes wird am Seriellen-Monitor Folgendes ausgegeben.

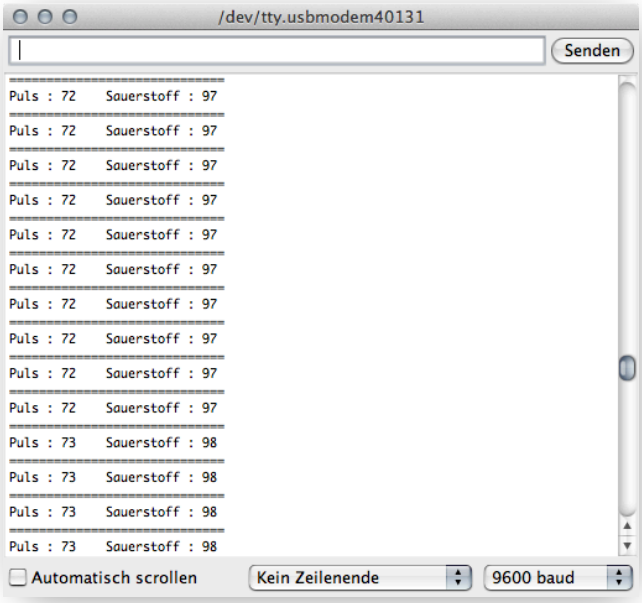


Abbildung 60: Ausgabe Serieller Monitor, Puls-Oxigen-Sensor

Überprüft werden die Werte anhand der oben aufgeführten Referenzwerte. Der am Sensor angeschlossene Proband ist 25 Jahre alt und treibt einmal die Woche Sport und fällt somit nicht in die Kategorie eines Athleten.

	Puls (bpm)	**Sauerstoffsättigung (%)**
Referenzwert ab 10 Jahre	60-100	96-98
Proband	72-73	97-98

Tabelle 3: Werte-Überprüfung Puls-Oxigen-Sensor

Die Werte des Probanden liegen innerhalb der Referenzwertbereiche, sodass die Aussage getroffen werden kann, dass der Sensor korrekt funktioniert und reale Werte zurückliefert.

5.2.4 Zusammenführen der Module

In diesem Abschnitt der Umsetzung wird das System, Arduino-Produzent, zusammengeführt und getestet. Die für dieses System relevanten Module wurden auf Funktionalität überprüft und die für die Umsetzung relevanten Programmcodes geschrieben.

Ziel der Umsetzung

Das System muss über die serielle Ausgabe die Werte der Sensoren getrennt durch ein Trennzeichen, zum Beispiel ein Semikolon, übermitteln damit diese vom Arduino-Client durch eine Funktion geparst und voneinander abgegrenzt werden können. Bevor bei jedem Durchlauf in der Loop-Schleife der Textstring mit den Sensorwerten übertragen wird, soll zudem ein eindeutiger Kopf (Header) gesendet werden, um auf Empfängerseite den Anfang der Nachricht eindeutig identifizieren zu können. Hierfür wird das Zeichen »H« verwendet, da der Rest der Nachricht aus Semikolons und natürlichen Zahlen besteht und somit gewiss ist, dass dieses Zeichen in der Textnachricht beziehungsweise Textstring kein weiteres Mal vorkommt.

Benötigte Module

- E-Health Sensor-Shield mit Puls-Oxigen-Sensor
- Accelerometer MMA8452
- Arduino-UNO R3
- XBee-Shield von Libelium und
- Zwei XBee-Module S1
- XBee Explorer USB
- USB-Kabel Typ-A auf Mini-B
- USB-Kabel Typ-A auf Typ-B
- Barrel-Jack-Adapter
- 9V Batterie

Benötigte Software

- Arduino 1.0.5
- CoolTerm

Benötigte Bibliotheken

- MMA8453_n0m1
- Wire
- PinChangeInt
- eHealth

Systemaufbau

Aufgrund der Pinbelegung (siehe Anhang J: Pinbelegung Arduino-UNO (Produzent)) und den Anschlüssen von Sensoren und der einzelnen Module wird zuerst auf den Arduino-UNO das Sensors-Shield aufgesteckt. Da das XBee-Shield von Libelium über eine Aussparung für den Anschluss des Puls-Oxigen-Sensors verfügt, kann nur dieses hierfür verwendet werden, verdeckt allerdings die Pins des Arduino-Boards für 3,3 V, GND, Reset, Vin und IOREF. Der 3,3 V Pin ist allerdings für den Accelerometer relevant und muss daher vor Anbringung des XBee-Shields damit verbunden werden. Die Pins D0 bis D7 und A0 bis A5 werden hingegen bis zum XBee-Shield durchgeschleust und sind darauf verwendbar. Die Sensoren werden demnach wie in den zuvor aufgeführten Testphasen und wie in folgender Abbildung angeschlossen.

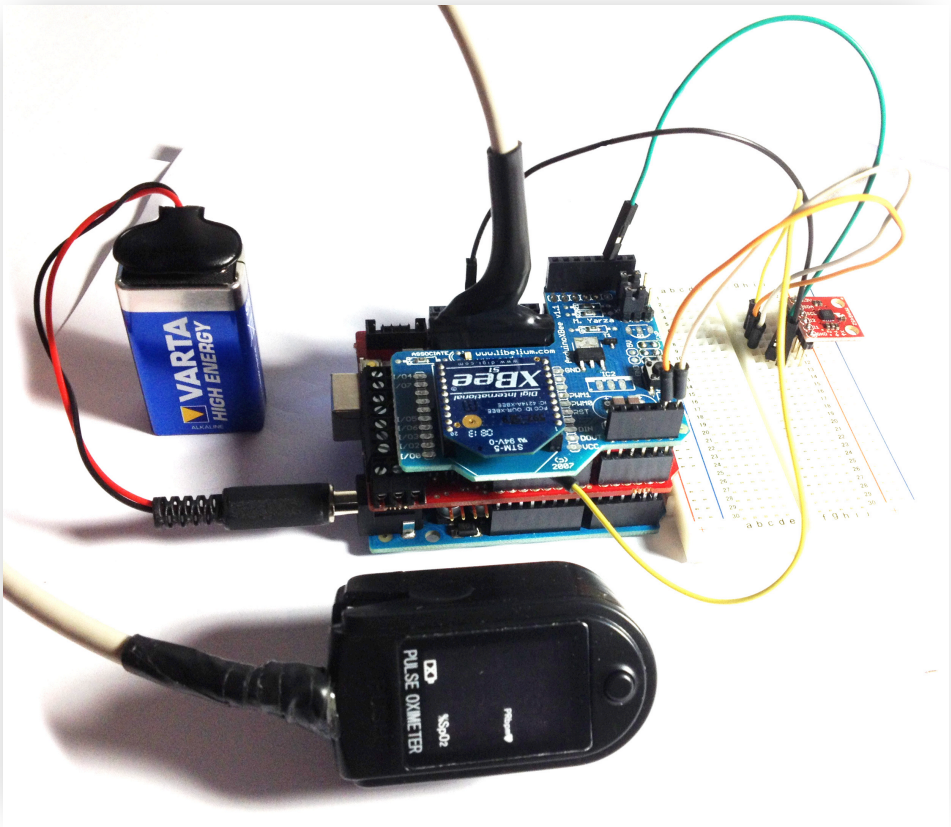


Abbildung 61: Systemaufbau Arduino-Produzent

In Abbildung 61 wird das Arduino-Board mit einer 9V-Batterie über einen Barrel-Jack-Adapter mit Strom versorgt. Bevor dieser Entwicklungsstand allerdings erreicht ist, muss das Arduino-Board noch programmiert werden und wird daher ohne externe Stromquelle über ein USB-Kabel mit Strom versorgt. In der ersten Testphase wird das Arduino-Board ohne XBee-Shield verwendet und die serielle Ausgabe über den Seriellen-Monitor ausgegeben. Dies dient zur Überprüfung des Programmcodes. Nach erfolgreicher Testphase wird das XBee-Shield mit vorkonfigurierten (siehe Kapitel 5.2.1) XBee-Modul aufgesteckt und anschließend das gesamte Arduino-Produzent-System getestet.

Umsetzung Programmcode

Der Programmcode muss einen Textstring an den seriellen Port senden, der wie folgt formatiert und genau diese Zeichenfolge übermittelt.

1. 'H' //Header – Kennzeichnet Beginn des Textstrings
2. ';' //Trennzeichen für Parser
3. (int) puls //Puls als Integer
4. ';' //Trennzeichen für Parser
5. (int) sauers //Sauerstoffsättigung als Integer
6. ';' //Trennzeichen für Parser
7. (int) zuck //Muskelzuckung '0'=Nein, '1'=Ja
8. '\r' // Carriage-Return

Aus den Programmcodes aus Kapitel 5.2.2 und 5.2.3 (Testen des Accelerometers und Testen des Puls-Oxigen-Sensors) entsteht folgender Code gemäß der Formatierung für den Textstring.

```
#include <Wire.h>
#include <PinChangeInt.h>
#include <eHealth.h>
#include <MMA8453_n0m1.h>

MMA8453_n0m1 accel;

int zaehler = 0;

void setup() {

  Serial.begin(9600);

  eHealth.initPulsioximeter();

  PCintPort::attachInterrupt(6,readPulsioximeter,RISING);

  accel.setI2CAddr(0x1D);
  accel.shakeMode(2,true,true,true,false,2);
}

void loop() {

  accel.update();

  // Ausgabe von Puls und Sauerstoffsättigung
  Serial.print("H");
  Serial.print(";");
  Serial.print(eHealth.getBPM());
```

```
  Serial.print(";");
  Serial.print(eHealth.getOxygenSaturation());
  Serial.print(";");

  // Ausgabe von ermittelten Zuckungen
  if(accel.shake()) {
    Serial.print("1");
  }
  else {
    Serial.print("0");
  }
  // Ende des Textstrings durch ein Carriage-Return kennzeichnen
  Serial.print('\r');
  delay(100);
}

void readPulsioximeter(){

  zaehler ++;

  if (zaehler == 50) {
    eHealth.readPulsioximeter();
    zaehler = 0;
  }
}
```

Quellcode 40: Gesamter Programmcode - Arduino-Produzent

Wird dieser Programmcode auf das Arduino-Board überspielt und die Serielle-Monitor-Anwendung gestartet, kann die serielle Ausgabe des Boards beobachtet und somit die Korrektheit des Textstrings wie die Messergebnisse überprüft werden.

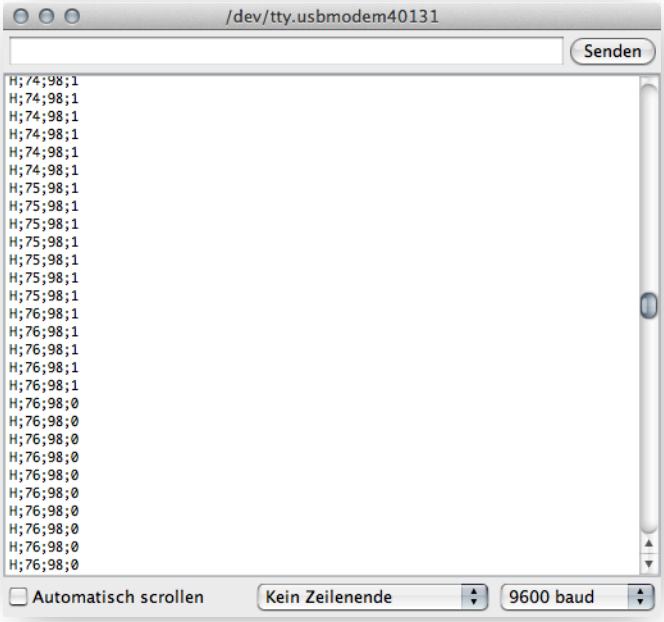


Abbildung 62: Testausgabe Serieller-Monitor Arduino-Produzent

Die Arduino-IDE gibt keinen Fehler beim Kompilieren des Sketches aus und aufgrund der richtigen Werte für Puls und Sauerstoffsättigung gemäß den Referenzwerten (zweiter und dritter Wert in

Abbildung 62) funktioniert der Puls-Oxigen-Sensor korrekt. Der vierte Wert (Accelerometer) im Textstring gibt an, ob Bewegungen stattfinden oder der Sensor sich im Ruhezustand befindet. Wird dieser in kleinen Intervallen in den verschiedenen Achsen bewegt, so wird auf dem Seriellen-Monitor der Wert »1« ausgegeben (siehe
Abbildung 62). Dadurch ist ebenfalls die Funktionalität dieses Sensors nachgewiesen und somit der gesamte Programmcode auf dessen Korrektheit überprüft.

Dem Anwender steht in diesem Code zudem ein Parameter zum Einstellen der Bewegungserkennungs-Sensibilität zur Verfügung. In der Funktion `shakeMode(2,true,true,true,false,2)` wird über den ersten Parameter die Sensibilität in einem Wertebereich zwischen 0 und 127 eingestellt. Je kleiner der Wert, desto früher und mit geringerer Beschleunigung wird eine Bewegung als solche erkannt und beim Aufrufen der Funktion `shake()`, `TRUE` zurückgeliefert.

Der Arduino-Sketch nimmt 8.708 Bytes Speicher in Anspruch. Der Arduino-UNO bietet ein Datenvolumen von 32.256 Bytes an. Daher ist es auch möglich ein Board zu verwenden, welches über wesentlich weniger Speicherplatz verfügt.

Hinzufügen des XBee-Shields mit Modul

Für die Kommunikation über Funk ist das XBee-Modul aus Kapitel 5.2.1 zu verwenden. Dieses wurde schon mit der Netzwerk-ID 2000 und einer Baudrate von 9600 konfiguriert. Diese Baudrate entspricht der im Programmcode eingestellten Rate und muss daher nicht neu konfiguriert oder eingestellt werden.

Die Kabel des Accelerometers (bis auf 3,3V Pin) zum Sensor-Shield müssen entfernt werden und anschließend das XBee-Shield auf das Sensor-Shield aufgesteckt werden. Darauffolgend muss der Accelerometer neu angeschlossen werden. Da die restlichen anzuschließenden Pins vom Arduino-Board bis auf das XBee-Shield durchgeschleust werden, können die Kabel vom Accelerometer auf dem XBee-Shield an den gleichen Pins wie zuvor angeschlossen werden (siehe Abbildung 61).

Systemtest

Für den Systemtest muss der Arduino-Produzent über die XBee-Module denselben Textstring wie in
Abbildung 62 ausgeben. Hierzu wird das Empfängermodul, bestehend aus XBee-Modul und XBee Explorer USB, an den Computer angeschlossen und anschließend die Software CoolTerm ausgeführt werden.

Der Arduino-Produzent wird nun auch mit einer externen 9V Stromquelle versorgt, sodass keine Verbindung zum Computer mehr nötig ist. Dadurch wird ebenfalls die autonome Lauffähigkeit des Systems aufgezeigt.

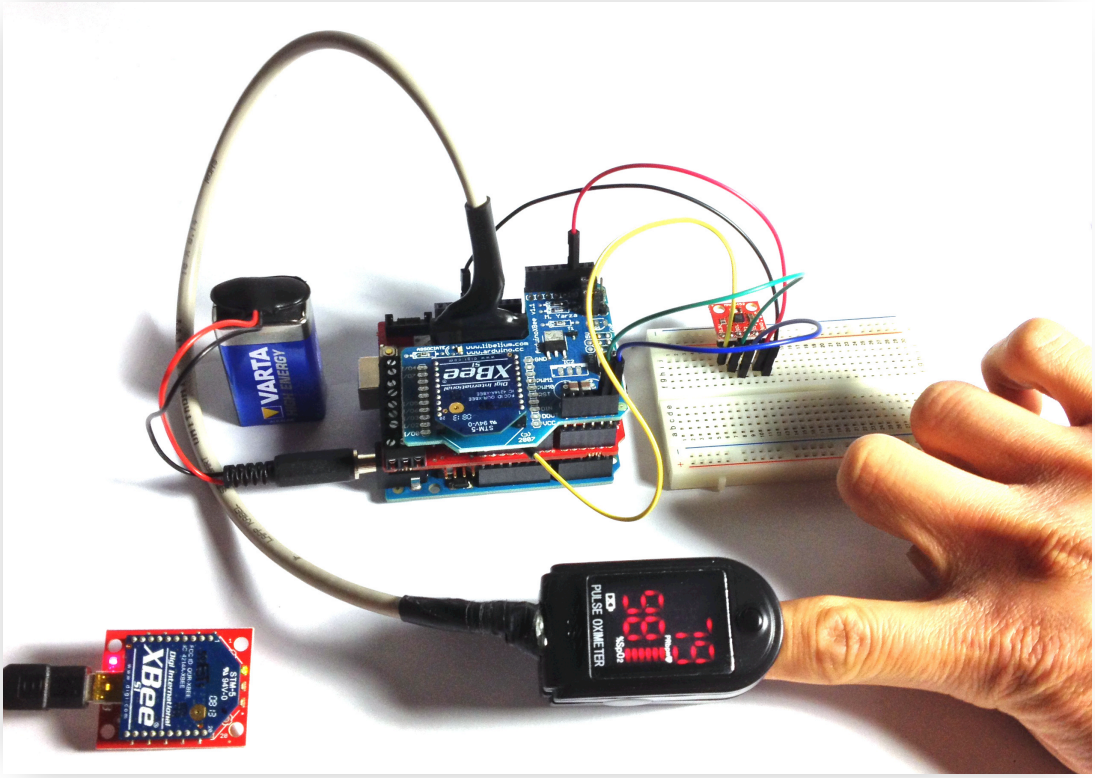


Abbildung 63: Systemtest Arduino-Produzent

Links unten in Abbildung 63 ist das XBee-Empfängermodul zu sehen, welches in serieller Datenübertragung den Textstring vom XBee-Modul des Arduino-Produzenten erhält. Dieses Modul wird im weiteren Verlauf der Umsetzungsphase beim Arduino-Client eingesetzt.

In CoolTerm wird nun der empfangene String in der Terminal-Anwendung ausgegeben und kann somit auf Korrektheit bezüglich des zuvor ausgeführten Tests, bei dem das Arduino über USB-Kabel mit dem Computer verbunden war, überprüft werden. Die Werte sind zwar nicht dieselben, allerdings ist bei Antreten des gleichen Probanden für den Testlauf die Wahrscheinlichkeit für nahezu identische Werte sehr hoch. Die Wahrscheinlichkeit, vergleichbare Werte zu erlangen wird dahingehend erhöht, wenn der Proband sich in ungefähr der gleichen vitalen Verfassung befindet, in der dieser den ersten Test durchgeführt hat. Das Ergebnis sieht in dieser Testphase wie folgt aus.

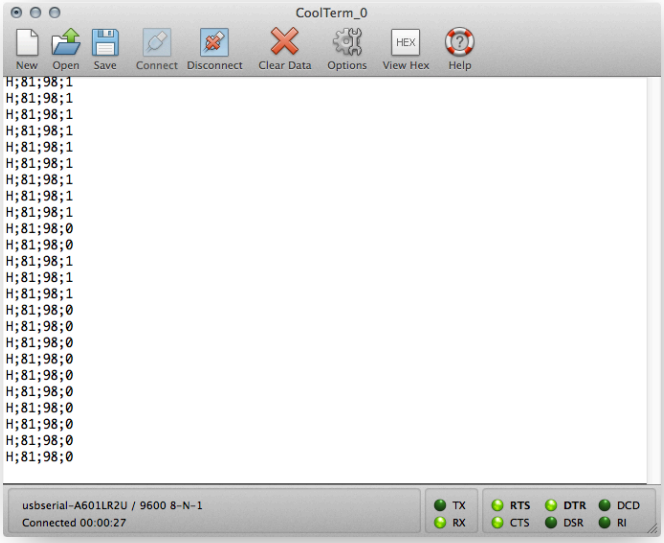


Abbildung 64: Ergebnis des Systemtests - Arduino-Produzent

Bewertung des Ergebnisses

Die Ergebnisse aus den beiden Testläufen mit und ohne XBee-Shield können nun miteinander verglichen werden.

	Puls (bpm)	Sauerstoffsättigung (%)
ohne XBee	74-76	98
mit XBee	81	98

Tabelle 4: Vergleichstabelle des Systemtests - Arduino Produzent

Die Werte für den Puls weichen nur gering ab und die Sauerstoffsättigung ist genau gleich hoch. Des Weiteren wurde beim Bewegen des Accelerometeres mit einer bestimmten Intensität der Wert »1« ausgegeben und somit die Funktionalität des Systems bestätigt. Unterschiedliche Ergebnisse in der Pulsoximetrie können durch verschiedene Faktoren hervorgerufen werden und lassen nicht auf Fehlfunktionen des Systems schließen. Da allerdings die Differenz zwischen den beiden Testergebnissen sehr gering ist und die Werte richtig formatiert am Empfängermodul ankommen, ist die Umsetzungsphase des Arduino-Produzenten somit beendet und der Client kann entsprechend dieses Textstrings programmiert und entwickelt werden.

5.3 Phase 2: Einrichten von Xively

In Kapitel 2.8.10 wurde ausführlich über den Funktionalitätsumfang des PaaS-Service Xively berichtet, der zur Visualisierung und Speicherung der Sensordaten bei der Implementierung eingesetzt wird. Aus diesem Kapitel sind auch die durch die Bibliothek integrierten Funktionen bekannt, welche in der folgenden Umsetzung eingesetzt werden.

Der Arduino-Produzent erfasst drei Sensorwerte. Darunter fallen der Sauerstoffgehalt, der Puls und der Wert, ob eine Bewegung beziehungsweise Muskelzuckung stattgefunden hat. Alle Werte müssen als Integer-Datentyp an die Xively-API übergeben werden.

Zuerst muss unter dem aktiven Xively-Account ein neues Entwicklungsgerät angelegt werden. Hierbei müssen ein Gerätename, eine Beschreibung und die Zugriffsrechte eingestellt werden. Durch diesen Schritt werden ein API-Key und eine Feed-ID generiert, welche später für den Arduino-Sketch des Client relevant werden. Anschließend müssen drei Channels[73] erstellt werden und eine entsprechende ID zum Zuordnen der im weiteren Verlauf der Thesis übermittelten Werte erstellt werden. Auch dieser Bezeichner wird im Arduino-Sketch für den Client benötigt. Zudem kann pro Channel ein Symbol oder Einheit gewählt werden, sodass der übertragende Wert in der korrekten Einheit angezeigt wird.

5.3.1 Konfiguration des Xively-Geräts

Folgende Konfigurationen wurden bei Xively vorgenommen oder durch das System generiert. Diese werden im weiteren Verlauf der Umsetzung benötigt.

Entwicklungsgerät anlegen/hinzufügen

- **Device Name** (siehe Abbildung 65-1)
  Epi-Log
- **Description**
  System zum Visualisieren von Sensorwerten einer Arduino Monitoring-Anwendung im Bereich der Epilepsie.
- **Privacy** (siehe Abbildung 65-1)
  Private Device (das Gerät ist nicht öffentlich sichtbar)

Generierte Keys und IDs

- **API Key** (siehe Abbildung 65-3)
  RFkq2DIIlYawGB4k1y0u2cFZDYeDQxk1DZ5uIEtm9j81ku2l
- **Feed ID** (siehe Abbildung 65-2)
  1178004072

Channels anlegen (siehe **Abbildung 65**-4)

Pro Sensorwert muss hierfür ein eigener Channel angelegt und ein eindeutiger Bezeichner vergeben werden.

- **Für Pulswert**
  Bezeichner(ID): Puls
  Symbol/Unit: bpm
- **Für Sauerstoffsättigung**
  Bezeichner(ID): Sauerstoffsaettigung
  Symbol/Unit: %
- **Für Zuckungen**
  Bezeichner(ID): Zucken
  Symbol/Unit: : 0=Nein; 1=Ja

[73] einen Kanal pro Sensor

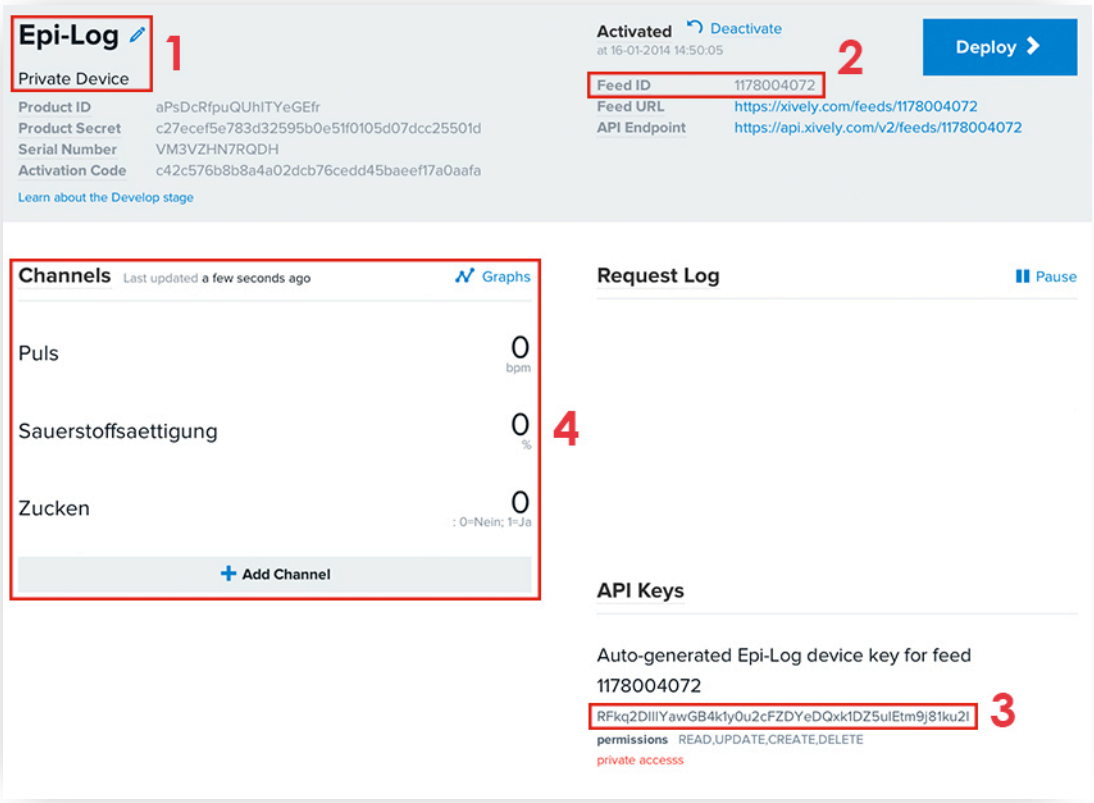


Abbildung 65: Xively-Workbench

Nach Einrichten des Entwicklungsgerätes gemäß den oben aufgeführten Parametern ist die Konfiguration für den Visualisierungsdienst abgeschlossen und kann nun in den Arduino-Client implementiert und getestet werden. Des Weiteren wurden in Kapitel 2.8.10 erste Testversuche mit Sensorwerten durchgeführt und eine Implementierung von Arduino aufgezeigt. Dahingehend wird die Kompatibilität zwischen Xively und Arduino als getestet angesehen und nicht weitergehend überprüft, sondern erst nach der Implementierung des Arduino-Clients im Systemtest veranschaulicht.

5.4 Phase 3: Umsetzung Arduino-Client

Dieses Kapitel befasst sich mit der Umsetzung des Arduino-Client, welcher die Daten vom Arduino-Produzent erfasst, verarbeitet und anschließend an Xively sendet. Das zu verwendende XBee-Modul zum Empfangen der Daten wurde in Kapitel 5.2.1 konfiguriert und getestet. Der Arduino-Client wird unter anderem als sogenanntes Gateway eingesetzt, welches Daten über den IEEE 802.15.4-Standard empfängt und diese anschließend über den IEEE 802.11-Standard an den Router übermittelt. Für dieses System sind auf dem Markt unter anderem Systeme vorhanden, welche diese Aufgabe übernehmen. Allerdings ist in Bezug zu dieser Thesis, in der Arduino in vernetzte Umgebungen implementiert werden soll, eine Arduino basierende Lösung von größerer Relevanz, zumal die Verarbeitung der Daten, wie auch das Hochladen auf den Server über den Client realisiert werden soll. Demnach ist es möglich, den Produzenten kleinstmöglich zu entwickeln und durch weniger Rechnerarbeit des Mikrocontrollers den Stromverbrauch des Arduino-Produzenten zu senken.

Die Umsetzung des Clients wird in zwei Phasen realisiert. In der Ersten wird das Arduino-Board um ein WiFi-Shield erweitert, welches die Kommunikation über den Router gewährleistet. Anschließend wird das vorkonfigurierte XBee-Modul angebracht und die Gateway-Funktionalität des Systems getestet. Diese Phase umfasst alle für dieses System einzusetzenden Hardwarekomponenten und daher muss in der zweiten Phase lediglich die Kommunikation zu Xively beziehungsweise zur JSON-API hergestellt werden. Diese Entwicklungsphase befasst sich zudem mit der Erstellung eines Programmcodes zum Einlesen der übermittelten Sensorwerte und der Zuordnung zu den jeweiligen Variablen. Dieses Verfahren wird »Parsen« genannt.

5.4.1 Testen des WiFi-Shields

Im Kapitel 2.6.4 wurden die Funktionen und Klassen der Ethernet- und WiFi-Bibliothek erläutert und werden bei der Umsetzung des Arduino-Clients für die Netzwerkkommunikation über ein WiFi-Shield verwendet. In diesem Kapitel wird das Arduino WiFi-Shield R3 auf Funktionalität überprüft.

Benötigte Hardware

- Arduino WiFi-Shield R3
- Arduino-UNO
- USB-Kabel Typ-A auf Typ-B

Benötigte Software

- Arduino 1.0.5

Benötigte Bibliotheken für Arduino

- WiFi
- SPI

Umsetzung

Für die Umsetzung des Testsystems muss das WiFi-Shield auf dem Arduino-UNO aufgesteckt und anschließend durch das USB-Kabel mit dem Computer verbunden werden. Die zu verwendenden Bibliotheken sind seit Arduino 1.0.5 in der Entwicklungsumgebung integriert und müssen nicht separat implementiert werden. Allerdings kann durch diese Version der Entwicklungsumgebung und das WiFi-Shield R3 Kompatibilitätsprobleme auftreten, welche allerdings durch ein Upgrade der Firmware, des HDG104-WiFi-Modul, behoben werden können.

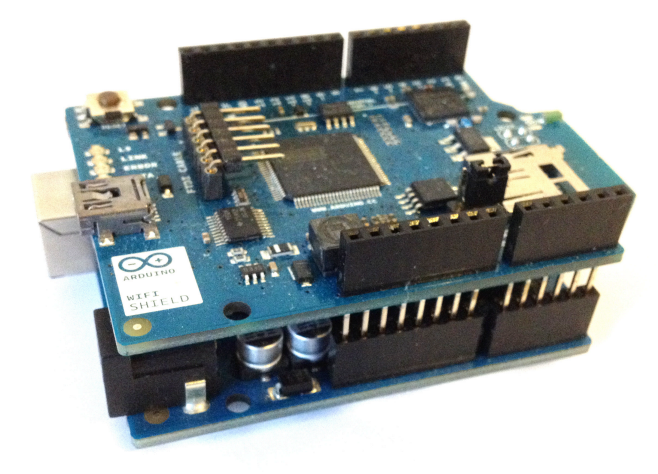


Abbildung 66: Arduino-UNO und WiFi-Shield

Zum Testen dieses Systems wird ein kleiner Programmcode geschrieben, welcher überprüft, ob der Arduino-UNO das Shield erkennt und dieses sich mit dem ausgewählten Netzwerk verbinden kann. Hierzu stellt Arduino durch die WiFi-Bibliothek die Funktion `status()` bereit, durch die der Verbindungsstatus abgefragt werden kann. Der Programmcode, um diese Funktionen zu testen sieht wie folgt aus.

```
#include <SPI.h>
#include <WiFi.h>


char ssid[] = "o2-WLAN57"; // Netzwerkname
char pass[] = "testPasswort"; // Passwort

int status = WL_IDLE_STATUS;

void setup() {

  Serial.begin(9600);

  // überprüfen ob Shield angesprochen werden kann:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi-Shield nicht vorhanden oder
    fehlerhaft!");
    // wenn nicht, dann nicht fortfahren:
    while(true);
  }
  else
    Serial.println("WiFi-Shield gefunden und einsatzbereit.");

  // versuchen sich mit Netzwerk zu verbinden:
  while (status != WL_CONNECTED) {
    Serial.print("Verbindungsaufbau mit Netzwerk: ");
    Serial.println(ssid);

    // Verbindung mit dem Netzwerk herstellen:
    status = WiFi.begin(ssid, pass);


    // 10 Sekunden für Verbindungsaufbau abwarten:
```

```
    delay(10000);
  }

  // Arduino mit WLAN verbunden, nun den Status ausgeben:
  wifiStatus();
}

void loop() {
  // für den Test irrelevant
}

void wifiStatus() {

  Serial.println("Verbindung erfolgreich hergestellt.");

  // ausgeben des Netzwerknamens:
  Serial.print("Netzwerkname: ");
  Serial.println(WiFi.SSID());

  // ausgeben der zugewiesenen IP-Adresse:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP-Adresse: ");
  Serial.println(ip);
}
```

Quellcode 41: Arduino-Sketch zum Testen des WiFi-Shields

In diesem Programmcode werden zuerst die beiden Bibliotheken, SPI und WiFi, inkludiert und anschließend der Netzwerkname (SSID) und das zugehörige Passwort festgelegt. Im Anschluss wird der WiFi-Status des Moduls in die Variable `status` geschrieben. Im Setup muss die serielle Kommunikation initialisiert werden, sodass nach Ausführen des Programmcodes die Statusmeldungen auf dem Seriellen-Monitor ausgegeben werden können. Über `WiFi.status()` wird anschließend überprüft, ob das Shield angesprochen werden kann und ob es schon verbunden ist. Wenn das Shield vorhanden ist, aber keine Verbindung zum Netzwerk besteht, so wird über `WiFi.begin(ssid, pass)` die Verbindung zum Netzwerk hergestellt und anschließend zehn Sekunden gewartet, damit die Verbindung aufgebaut werden kann. Daraufhin wird die Funktion `wifiStatus()` aufgerufen, in welcher der Netzwerkname und die zugewiesene, lokal verfügbare IP-Adresse ausgegeben werden.

Auf dem Seriellen-Monitor ist in dem Netzwerk des Autors dieser Thesis dabei Folgendes zu sehen.

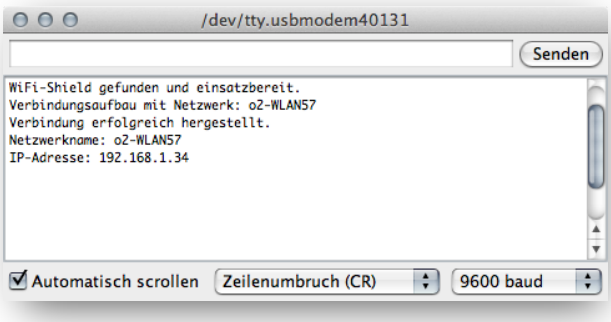
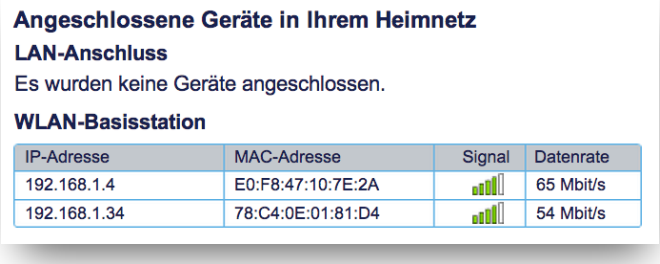


Abbildung 67: Ausgabe WiFi-Status am Seriellen-Monitor

Anhand dieser seriellen Ausgabe ist zu erkennen, dass der Arduino das WiFi-Shield erkennt und eine Verbindung zum Netzwerk herstellen kann. Dies kann anschließend gegengeprüft werden, indem die IP-Adresse mit den Adressen im Administrationsbereich des Routers verglichen wird.



Angeschlossene Geräte in Ihrem Heimnetz

LAN-Anschluss

Es wurden keine Geräte angeschlossen.

WLAN-Basisstation

IP-Adresse	MAC-Adresse	Signal	Datenrate
192.168.1.4	E0:F8:47:10:7E:2A		65 Mbit/s
192.168.1.34	78:C4:0E:01:81:D4		54 Mbit/s

Abbildung 68: Administrationsbereich des Routers im Netzwerk: o2-WLAN57

In Abbildung 68 ist die IP-Adresse des Arduino aufgeführt, wonach die Verbindung zum Heimnetzwerk erfolgreich hergestellt und die Funktionalität des WiFi-Shields überprüft und für funktionsfähig erklärt wird.

5.4.2 Hinzufügen des XBee-Moduls

Das XBee-Modul wurde in Kapitel 5.2.1 konfiguriert und getestet und kann in dieser Phase der Umsetzung auf das WiFi-Shield aufgesteckt werden. Auch hierbei muss überprüft werden, welche Pins des Arduinos bisher durch das WiFi-Shield belegt wurden und dem XBee-Shield noch zur Verfügung stehen. In Anhang K: Pinbelegung Arduino-UNO (Client), sind die durch die Shields belegten Pins aufgeführt und müssen auf Überlappung beziehungsweise Doppelbelegung überprüft werden. Durch das WiFi-Shield wird Pin D7 für den Handshake zwischen Shield und Arduino verwendet, Pin D10 für das HDG104-WiFi-Modul und Pin D11 bis D13 für die Kommunikation über SPI. Das XBee-Shield mit Modul benötigt Pin D0 und D1 für die serielle Kommunikation. Demnach ist kein Pin des Arduino doppelt belegt und das XBee-Shield kann entsprechend auf das WiFi-Shield aufgesteckt werden. Bei dem XBee-Shield handelt es sich um eine Version von Sparkfun und wird ohne Header-Leiste ausgeliefert. Aufgrund dessen müssen diese zuvor angebracht beziehungsweise angelötet werden.

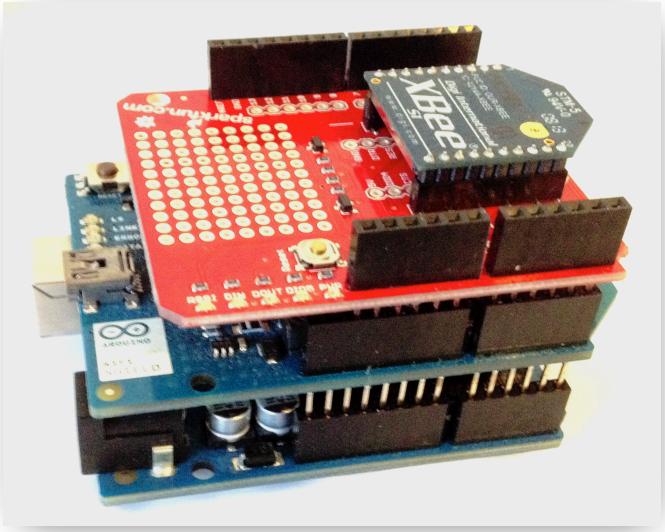


Abbildung 69: Arduino-Client Hardware

5.4.3 Verarbeitung des Textstrings

Zu diesem Zeitpunkt der Implementierung von Arduino als Prototyp für eine Monitoring-Anwendung sind die Hardwarekomponenten erfolgreich zusammengeführt und getestet. Allerdings muss der vom Arduino-Produzent übermittelte Textstring noch verarbeitet und die Werte an Xively übertragen werden.

In diesem Kapitel wird eine Funktion geschrieben, welche den eintreffenden Textstring in einen Puffer schreibt, anschließend parst und in drei für die Sensorwerte vorgesehenen Variablen ablegt.

Der Textstring hat fünf Eigenschaften, die beim Parsen berücksichtigt werden müssen.

1. Jeder String beginnt mit einem »H«.
2. Delimiter oder Trennzeichen ist ein Semikolon.
3. Die Sensorwerte müssen als Integer abgelegt werden.
4. Der Puls und die Sauerstoffsättigung können eine bis drei Ziffern umfassen, der Bewegungsparameter hingegen nur eine.
5. Der Textstring endet mit einem Carriage-Return.

Daraus geht hervor, dass die kürzeste übermittelte Textnachricht acht Zeichen und somit auch acht Bytes umfasst (zum Beispiel H;0;0;0\r). Die Längste hingegen bis zu 12 Bytes (zum Beispiel: H;999;999;1\r).

Umsetzung des Parsers

Der eintreffende Textstring muss Byte für Byte eingelesen und in einen Puffer mit mindestens 12 Bytes abgelegt werden. Dieser soll sobald ein Carriage-Return (Ende des Textstrings) erreicht ist, geparst werden und hierbei die Werte bis zu einem Semikolon in jeweils separaten Variablen ablegen. Anschließend soll der Puffer gelöscht und neu eingelesen werden.

Für die serielle Kommunikation wurden im früheren Verlauf der Thesis für Testausgaben schon Funktionen wie `Serial.begin()` oder `Serial.print()` angewandt. Zum Einlesen der Daten stellt die Arduino-Entwicklungsumgebung weitere Funktionen zur Verfügung.

- **Verfügbar unter()**
  Prüft den Eingangspuffer ob an der seriellen Schnittstelle Daten eintreffen und gibt die Anzahl dieser Bytes zurück.
- **read()**
  Liest das nächste Zeichen aus dem Eingangspuffer. [8]

Es gibt noch weitaus mehr Funktionen, allerdings sind diese beiden für die Umsetzung des Arduino-Clients relevant.

Über `#define PUFFER_SIZE 13` wird zu Beginn des Programmcodes dem konstanten Wert 13 der Name »PUFFER_SIZE« zugewiesen. Dies hat den Vorteil, dass der Compiler alle Verweise zu dieser Konstante mit dem definierten Wert ersetzt und somit keinen Programmspeicherplatz benötigt. Der Wert umfasst die zuvor ermittelten 12 Bytes plus einen weiteren als Platzhalter für die maximale Länge eines Textstrings, der vom Arduino-Produzent übermittelt wird.

Noch vor dem Setup wird ein Array (`puffer[]`) vom Character-Datentyp mit der Anzahl von 13 Elementen, durch `PUFFER_SIZE` festgelegt, deklariert, um im weiteren Verlauf diesen mit den eintreffenden Bytes zu füllen. Des Weiteren wird eine Variable (`eintreffendeBytes`) ebenfalls vom Character-Datentyp deklariert und definiert, um das momentan eintreffende Byte zu speichern. Zusätzlich wird noch ein Zähler (`bytesZaehler`) vom Datentyp Integer initialisiert, der die einzelnen Stellen (Bytes) im Array durchläuft. Um die geparsten Werte Variablen zuordnen zu können, müssen auch diese vorab initialisiert werden. Hierbei handelt es sich um drei Variablen vom Integer-Datentyp (`pulsWert`, `sauersWert`, `bewegWert`), die jeweils mit dem Wert »0« initialisiert werden. Das hat den Vorteil, dass wenn zu Programmstart noch keine Werte vom Arduino-Produzenten übermittelt werden und zu diesem Zeitpunkt schon eine Verbindung zu Xively steht, immer der Wert »0« übertragen wird und somit keine Fehler beim serverseitigen Empfangen entstehen.

Die Funktion `leseSerial()` übernimmt das Einlesen der über die serielle Schnittstelle übermittelten Bytes und deren Verarbeitung.

```
void leseSerial() {

  while(Serial.Verfügbar unter()) {

    eintreffendeBytes = Serial.read();

    if(eintreffendeBytes != '\r') {
      puffer[bytesZaehler++] = eintreffendeBytes;
    }
    else {
      if(puffer[0]!='H') {
        bytesZaehler = 0;
        memset(puffer, 0, PUFFER_SIZE);
      }
```

```
        else {
          puffer[bytesZaehler] = '&#092;&#048;';
          char *header = strtok(puffer, ";");
          char *puls = strtok(NULL, ";");
          char *sauer = strtok(NULL, ";");
          char *bewegung = strtok(NULL, ";");

          pulsWert = atoi(puls);
          sauersWert = atoi(sauer);
          bewegWert = atoi(bewegung);

          bytesZaehler = 0;
          memset(puffer, 0, PUFFER_SIZE);
        }
      }
    }
  }
```

Quellcode 42: Funktion zum Einlesen der Bytes und Parsen

Diese Funktion soll aus der Loop-Schleife aufgerufen werden, sodass nach jedem Übertragen der Daten auf den Server, die Funktion neu ausgeführt und neue Daten ermittelt werden. Nach dem Funktionsaufruf wird in eine `while`-Schleife gewechselt, solange Bytes im Eingangspuffer liegen. Anschließend wird über `Serial.read()` das erste Byte in die Variable `eintreffendeBytes` geschrieben und somit aus dem Eingangspuffer gelöscht. Anschließend wird überprüft, ob es sich bei diesem Byte um das Carriage-Return handelt, welches das Ende des übertragenden Textstrings kennzeichnet. Ist dies nicht der Fall, so wird dieses Byte an die erste Stelle im Puffer geschrieben. Dieser Vorgang wiederholt sich so lange bis ein Carriage-Return im aktuellen Byte liegt. Dann ist die Bedingung nichtig und es wird in den `else`-Fall gewechselt. Hier wird überprüft, ob das erste Zeichen ein »H« ist, welches den Beginn des Textstrings beschreibt. Ist das nicht der Fall, so wird der Zähler für den Array wieder auf »0« gesetzt und der Puffer gelöscht beziehungsweise jeder Wert durch eine »0« ersetzt. Ist hingegen die zuvor erwähnte Bedingung wahr und das erste Zeichen im Puffer ein »H«, dann wird an die letzte Stelle des Puffers »NULL« beziehungsweise »\0« geschrieben, was das Ende des Arrays angibt.

Anschließend wird über die Funktion `strtok(puffer, ";")` der Textstring anhand eines Semikolons zerteilt und die einzelnen Abschnitte herausgelesen. In dem ersten Funktionsaufruf muss `strtok()` mit einem Textstring initialisiert werden und liefert anschließend den ersten Abschnitt zurück. In diesem Fall ist das immer das Zeichen »H«, welches in die Variable `header` geschrieben und nicht weiter benötigt wird. Bei den Folgeaufrufen wird kein String mehr übergeben, sondern der NULL-Wert, da `strtok()` bereits initialisiert ist und einen Zeiger auf den String gespeichert hat. Die folgenden Variablen (`puls, sauer, bewegung`) sind Zeiger, welche lediglich den Rückgabewert der Funktion abfangen und auf das erste Zeichen des jeweiligen Abschnittes zeigen. Der Delimiter oder das Trennzeichen wird hierbei mit NULL überschrieben. Das Ende des Textstrings ist erreicht, wenn `strtok()` den NULL-Zeiger zurückliefert. Aufgrund dessen wurde an die letzte Stelle im Puffer das Zeichen »\0« geschrieben, sodass das Ende für diese Funktion erkennbar ist.

Nachdem die Zeichen für die jeweiligen Werte geparst wurden, müssen diese noch in Integer-Werte gewandelt werden. Hierzu wird die Funktion `atoi()` verwendet. Diese liefert als Funktionswert die ausgelesene, ganze Zahl zurück. Tritt ein Zeichen auf, welches

nicht als ganze Zahl erkannt wird, so bricht diese Funktion zudem ab und es kann nicht vorkommen, dass durch einen Fehler bei der Zerlegung der Textnachricht sonstige Zeichen zurückgeliefert werden. Diese Rückgabewerte werden in diesem Zusammenhang in die jeweiligen Integer-Variablen geschrieben und anschließend der Zähler für den Array genullt und alle im Puffer befindlichen Zahlen mit einer »0« überschrieben.

Testen der Funktion

Für einen Test muss die eben beschriebene Funktion außerhalb der Loop-Schleife in einen Arduino-Sketch eingefügt werden. Anschließend muss im Setup die serielle Kommunikation mit einer Baudrate von 9600 initialisiert werden und in der Loop-Schleife über `Serial.print()` die Werte aus den Variablen (`pulsWert`, `sauersWert`, `bewegWert`) ausgegeben werden.

Benötigte Hardware

- Arduino-Produzent
- 9 V Netzteil
- Arduino-UNO
- XBee-Shield von Sparkfun
- XBee-Modul S1
- USB-Kabel Typ-A auf Typ-B

Benötigte Software

- Arduino 1.0.5

Umsetzung

Bei diesem Test handelt es sich um eine für die prototypische Implementierung umgesetzte Funktion, welche in diesem Zusammenhang auf Funktionalität überprüft werden muss. Der Arduino-Produzent wird hierbei durch ein 9V Netzteil mit Strom versorgt und sendet realistische Daten, die von den Sensoren erfasst werden. Auf Empfänger- beziehungsweise Client-Seite werden diese gemäß der oberen Funktion geparst und anschließend über die serielle Schnittstelle ausgegeben. Hierzu wird auf den Arduino-UNO der Programmcode geladen und anschließend das XBee-Shield mit Modul aufgesteckt. Das USB-Kabel bleibt weiterhin mit dem System verbunden, sodass die serielle Ausgabe über den Seriellen-Monitor überprüft werden kann. Werden die Werte als ganze, natürliche Zahlen ausgegeben, so ist die Funktionalität dieser Funktion bewiesen und überprüft.

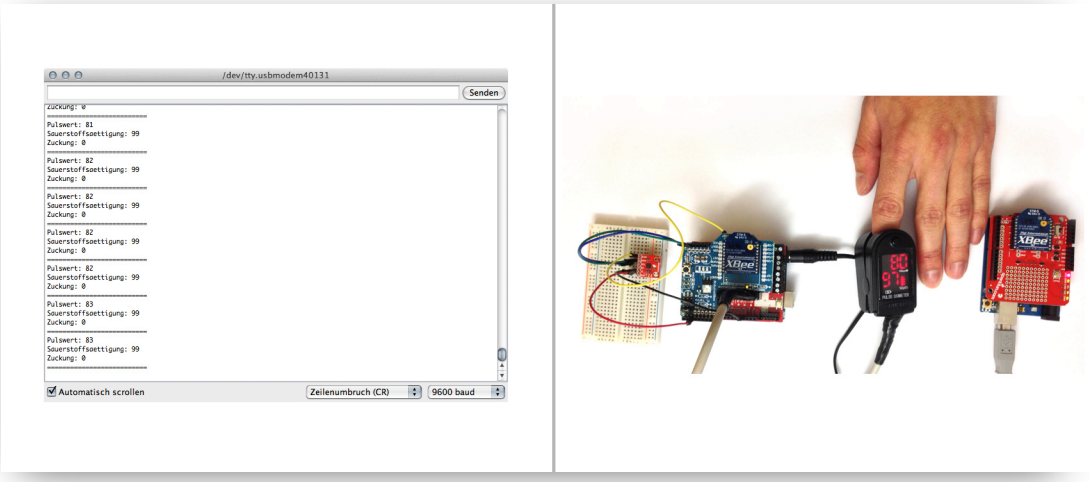


Abbildung 70: Funktionstest des Parsers

In dieser Abbildung und durch diesen Test ist eindeutig zu erkennen, wie der eintreffende Textstring gemäß den Anforderungen korrekt geparst wird und somit kann die Funktion im Arduino-Client implementiert werden.

5.4.4 Einbinden von Xively in Arduino-Client

Im Kapitel 5.3 wurde der Xively-Feed eingerichtet und muss nun noch in Arduino implementiert werden. In diesem Zusammenhang wird der zu einem früheren Zeitpunkt in der Umsetzungsphase geschrieben WiFi-Programmcode wie auch der direkt zuvor entwickelte Parser-Code in dem Arduino-Sketch zusammengeführt und um die Funktionalität zum Übermitteln der Daten an Xively erweitert.

Benötigte Hardware

- Arduino-Produzent
- Arduino-Client
- USB-Kabel Typ-A auf Typ-B

Benötigte Software

- Webbrowser für Xively-Anwendung
- Arduino 1.0.5

Benötigte Bibliotheken

- WiFi
- SPI
- **Xively_Arduino-Library**
  Diese Bibliothek ermöglicht mit wenig selbstgeschriebenen Programmcode, Xively in einen eigenen Arduino-Sketch zu implementieren. Wird auf folgender Seite zum Download bereitgestellt: *https://github.com/xively/xively_arduino*.

- **HttpClient-Library**
  Eine Bibliothek um mit geringem Aufwand eine Interaktion zwischen Arduino und einem Web-Server herzustellen. Nicht speziell für Xively entwickelt, wird allerdings von der Xively_Arduino-Bibliothek vorausgesetzt. Der Download wird auf folgender Seite angeboten: *https://github.com/amcewen/HttpClient*.

Benötigte Xively-Parameter (aus Kapitel 5.3.1)

- **API Key**
  RFkq2DIIlYawGB4k1y0u2cFZDYeDQxk1DZ5uIEtm9j81ku2l
- **Feed ID**
  1178004074
- **Pulswert ID**
  Puls
- **Sauerstoffsättigung ID**
  Sauerstoffsaettigung
- **Zuckungen ID**
  Zucken

Umsetzung

Diese Umsetzung umfasst alle in diesem Kapitel erwähnten Hardwarekomponenten wie auch Funktionen und schließt die Implementierung des Arduino in den Client ab. Im Anschluss an dieses Kapitel wird ein Systemtest über eine gewisse Dauer durchgeführt, um die Stabilität des gesamten Systems zu überprüfen.

Um allerdings die WiFi-, XBee- und Xively-Funktionalität in einem Programmcode zusammenzuführen, werden die vier oben ausgewiesenen Bibliotheken benötigt. Die SPI- und WiFi-Bibliothek sind wie schon erwähnt in der Entwicklungsumgebung integriert. Allerdings muss die Xively- wie auch HttpClient-Bibliothek heruntergeladen und im Bibliotheks-Verzeichnis von Arduino abgelegt werden. Anschließend wird ein neuer Arduino-Sketch geöffnet und diese vier Bibliotheken inkludiert.

Auf die Implementierung der Parser-Funktion wird in diesem Zusammenhang nicht näher eingegangen, da diese aus dem vorherigen Kapitel bekannt ist. Lediglich der Funktionsaufruf wird im folgenden Quellcode aufgezeigt.

Für eine Verbindung zum Router über das WiFi-Shield muss der Netzwerkname wie auch das zugehörige Passwort in den Programmcode geschrieben werden. Vor dem Setup muss zudem ein String deklariert und initialisiert werden, welcher den API-Key von Xively beinhaltet. Anschließend werden drei Strings für die jeweilige ID für Puls, Sauerstoffsättigung und Zuckungen initialisiert. Im Anschluss wird ein `XivelyDatastream` Objekt-Array mit drei Objekten für die jeweiligen Sensorwerte erzeugt. Hierzu werden der Identifier und der Datentyp zum Anlegen benötigt. Der Datentyp wird über `DATASTREAM_INT` angegeben. Alle Werte sollen als Integer beziehungsweise ganze, natürliche Zahlen übertragen werden. Dieser Datenstream wird darauffolgend in ein `XivelyFeed` Objekt gepackt, in dem die Feed ID und die Anzahl der Datenstreams vermerkt werden. Anschließend wird ein `XivelyClient` angelegt, welcher im weiteren Verlauf den HTTP-Request sendet.

Im Setup wird lediglich das WiFi-Shield initialisiert und eine Verbindung zum Router aufgebaut. In der Loop-Schleife wird zu Beginn die Funktion `leseSerial()` ausgeführt. Diese erfasst, wie im Kapitel zuvor beschrieben, die Sensordaten und weist diese den jeweiligen Variablen zu, sodass diese an Xively übergeben werden können. Dadurch können anschließend den einzelnen Datenstreams die Werte der Variablen übergeben werden. Dies wird über eine Funktion der Xively-Bibliothek bereitgestellt und durch `datastream[0].setInt(pulsWert)` für alle drei Variablen (`pulsWert`, `sauersWert`, `bewegWert`) durchgeführt. Im Anschluss muss durch eine `put()`-Funktion das Feed-Objekt auf den Xively-Account beziehungsweise an die JSON-API per HTTP-Request gesandt werden. Daraufhin wird noch für die Testphase der HTTP-Statuscode ausgeben, um zu überprüfen, ob der Datentransfer erfolgreich war.

Im folgenden Quellcode sind alle zuvor erwähnten Funktionen, bis auf die `leseSerial()`-Funktion, enthalten, die allerdings im Anschluss an die Loop-Schleife implementiert und auf das Arduino-Board geladen wurde.

```
#include <SPI.h>
#include <WiFi.h>
#include <HttpClient.h>
#include <Xively.h>

char ssid[] = "o2-WLAN57";
char pass[] = "testPasswort";

int status = WL_IDLE_STATUS;

char xivelyKey[] =
"RFkq2DIIlYawGB4k1y0u2cFZDYeDQxk1DZ5uIEtm9j81ku2l";

char pulsId[] = "Puls";
char soId[] = "Sauerstoffsaettigung";
char zuckId[] = "Zucken";
XivelyDatastream datastreams[] = {
  XivelyDatastream(pulsId, strlen(pulsId), DATASTREAM_INT),
  XivelyDatastream(soId, strlen(soId), DATASTREAM_INT),
  XivelyDatastream(zuckId, strlen(zuckId), DATASTREAM_INT),
};

XivelyFeed feed(1178004072, datastreams, 3);

WiFiClient client;
XivelyClient xivelyclient(client);

void setup() {

  Serial.begin(9600);

  while ( status != WL_CONNECTED) {
    status = WiFi.begin(ssid, pass);
    delay(100);
  }
}

void loop() {
  leseSerial();

  datastreams[0].setInt(pulsWert);
```

```
  datastreams[1].setInt(sauersWert);

  datastreams[2].setInt(bewegWert);

  int ret = xivelyclient.put(feed, xivelyKey);
  Serial.println(ret);
  delay(100);
}
```

Quellcode 43: Programmcode des Arduino-Produzenten, ohne leseSerial()-Funktion

Durch diesen Programmcode, ergänzt um die `leseSerial()`-Funktion, ist auch der Arduino-Produzent fertiggestellt und kann getestet werden.

Testen des Arduino-Clients

Hierzu werden beide Systeme benötigt und müssen über den aus den einzelnen Kapiteln resultierenden Programmcode verfügen. Der Arduino-Produzent wird mit einem 9V Netzteil und der Arduino-Client über das USB-Kabel mit Strom versorgt. Dadurch kann beim Client auf dem Seriellen-Monitor der HTTP-Response von Xively ausgegeben werden.

Sind beide wie folgt angeschlossen, werden unter dem Entwicklungsgerät auf Xively die Ausgaben und der Request-Log überprüft. Der Request-Log zeigt die einzelnen HTTP-Requests an. Im Request-Body können die im JSON-Format vorliegenden Daten auf deren Korrektheit überprüft werden.

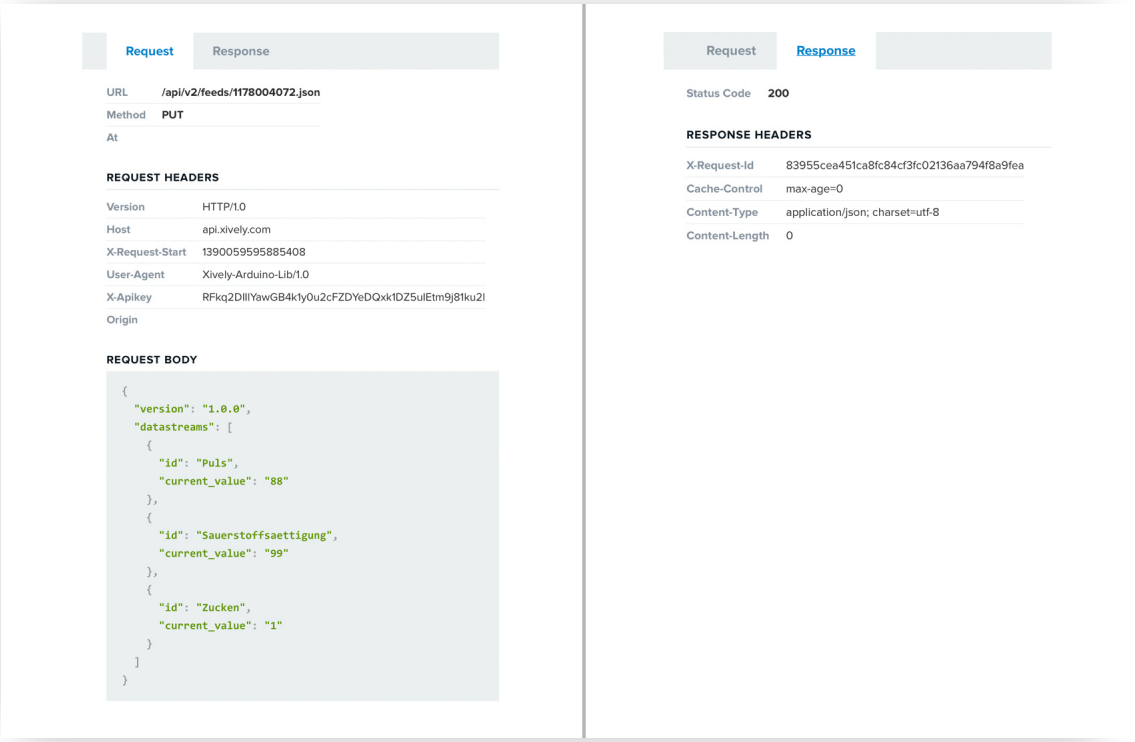


Abbildung 71: Screenshot Xively HTTP-Request und Response

In der rechten Hälfte des Screenshots ist ein HTTP-Request gezeigt. Dieser wird auf Korrektheit im Request-Body überprüft. Hierbei ist zu sehen, dass die Bezeichner korrekt sind und die Werte als ganze, natürliche Zahlen übermittelt werden. Der HTTP-Response auf der rechten Hälfte des Bildes gibt an, dass der HTTP-Statuscode »200« übermittelt wurde und somit alles in Ordnung war.

Die einzelnen übermittelten Werte in Bezug zu den Referenzwerten und Bewegungen am Accelerometer werden in den drei Graphen überprüft.

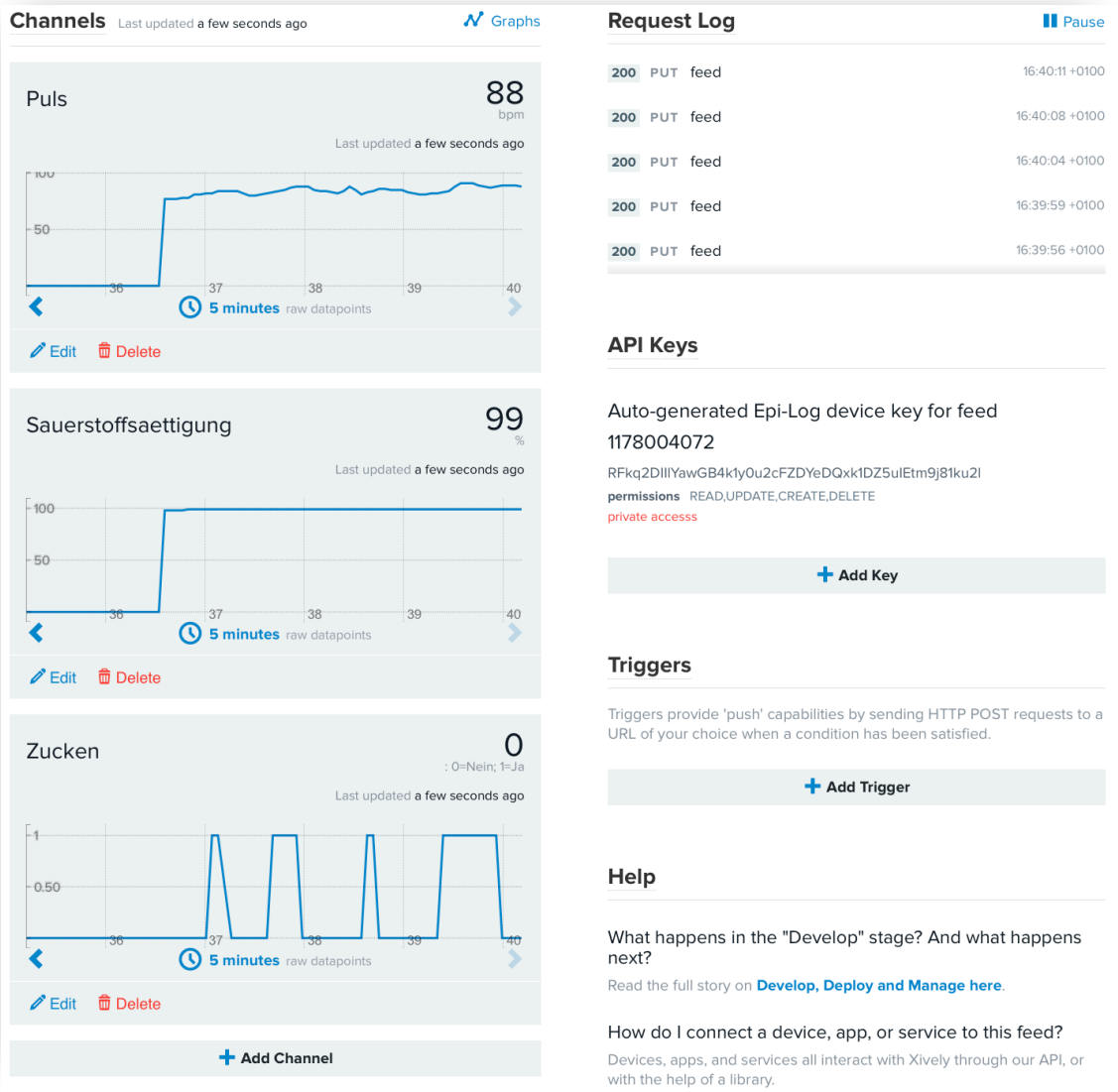


Abbildung 72: Screenshot Xively-Workbench mit finalem Prototyp

Diese drei Graphen für Puls, Sauerstoffsättigung und Zuckungen lassen sich dahingehend überprüfen, dass der Puls in der vierminütigen Testphase zwischen 74 und 92 Schlägen pro Minuten liegt und der Sauerstoffgehalt anfangs bei 98 und anschließend bei 99 Prozent. Diese Werte entsprechen den im Kapitel 5.2.3 Referenzwerten. Der Accelerometer kann getestet werden, indem ab einer gewissen Intensität bei der Bewegung des Sensors der Wert auf »1« steigt und anschließend wieder auf »0« fällt. Dabei fällt auf, dass die Werte nur alle drei bis vier Sekunden übermittelt werden beziehungsweise serverseitig Verarbeitungszeit benötigen. Für eine prototypische Implementierung stellt dies allerdings kein Problem dar. Bei einem marktreifen Produkt wäre eine derartige Latenz allerdings

auch vertretbar, da die Abbildung der Werte für den behandelnden Arzt von großer Relevanz wären und diese somit nicht in der Schlafphase überprüft, sondern zeitdiskret analysiert werden würden.

Somit ist die prototypische Implementierung von Arduino in eine Monitoring-Anwendung abgeschlossen und deren Umsetzbarkeit bewiesen. Allerdings wird im folgenden Kapitel ein größerer Systemtest durchgeführt und der finale Prototyp bezüglich der erhobenen Anforderungen überprüft.

5.5 Phase 4: Finaler Systemtest

In diesem Kapitel wird der finale Systemtest durchgeführt, bei dem die Arduino-Systeme, konfiguriert und programmiert wie im letzten Kapitel, bei einer Laufzeit von einer Stunde auf Funktionalität und Stabilität überprüft werden. Anschließend wird geprüft, ob die in der Anforderungsanalyse ermittelten Anforderungen durch dieses System erfüllt werden. Die Testumgebung kann hierbei allerdings nicht der Produktivumgebung entsprechen, da die Monitoring-Anwendung nicht an einem Patienten angebracht werden kann, da es sich um einen Prototyp ohne CE-Zertifizierung und Zulassung gemäß den Richtlinien 93/42/EWG für Medizinprodukte handelt.

Die Funktionalität der einzelnen Sensoren und Module wurde in den vorhergehenden Kapiteln in sogenannten Unit-Tests durchgeführt und müssen daher kein weiteres Mal überprüft werden. Daher wird der finale Prototyp in zwei Phasen getestet. In der ersten Phase wird das System hauptsächlich auf Stabilität getestet, indem die Sensoren entsprechend angesprochen und manipuliert werden. Die zweite Phase untersucht das Verhalten des Systems bei Ausfall von Hardwarekomponenten oder nicht vorhergesehenen Ereignissen, welche Fehlfunktionen verursachen können.

Der Prototyp und Programmcode entspricht dem System im vorherigen Kapitel und wird unverändert in diesen Tests eingesetzt.

5.5.1 Phase 1: Stabilitätstest

Überprüfung des Systems auf Stabilität bei einer Testdauer von einer Stunde. Der Proband führt Bewegungen am Accelerometer durch und ist am Puls-Oxigen-Sensor angeschlossen.

Pulsmessung

Der Pulsschlag wird aufgrund der am Accelerometer ausgeübten Bewegungen dahingehend verändert, dass der Proband für diese simulierten Zuckungen einen gewissen Aufwand betreiben muss und somit eine Anstrengung ausgeübt wird, wodurch sich der Pulsschlag erhöht.

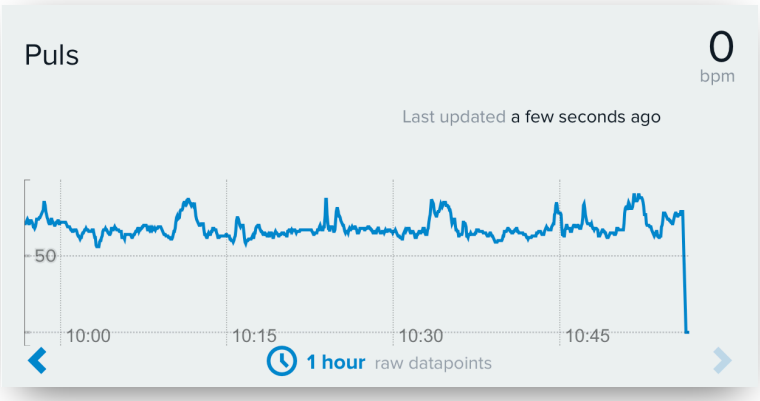


Abbildung 73: Xively Pulsmessung im Systemtest

Sauerstoffsättigungsmessung

Für die Sauerstoffsättigung sind Referenzwerte vorhanden, wonach überprüft werden kann, ob die gemessenen Werte sich an diesen orientieren. Des Weiteren ist wichtig zu überprüfen, ob bei einer Testdauer von einer Stunde der Wert sich außerhalb der Referenzwerte befindet oder sogenannte Ausreißer vorhanden sind, welche auf Fehlfunktionen des Systems hinweisen könnten.

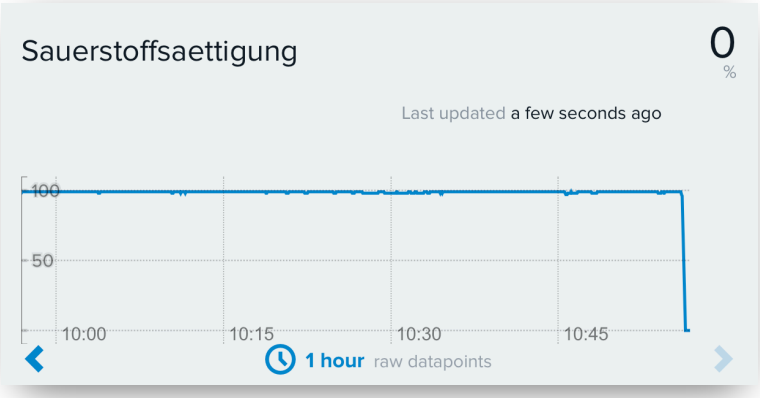


Abbildung 74: Xively Sauerstoffsättigungsmessung im Systemtest

Bewegungsmessung

In diesem Test wird überprüft, ob eine Zuckung beziehungsweise Muskelzuckung nur dann erkannt wird, wenn diese auch gewollt hervorgerufen wird. Durch kontinuierliches Ausüben von normalen, dem Schlaf nachempfundenen Bewegungsabläufen, darf hingegen kein Zucken als solches erfasst werden.

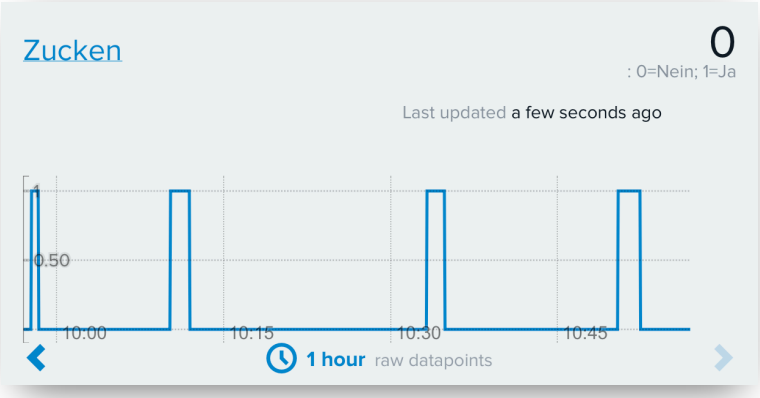


Abbildung 75: Xively Bewegungsmessung im Systemtest

Ergebnis

In dieser Testphase haben die Sensoren wie gefordert gearbeitet und gemäß den Aktivitäten des Probanden die Veränderungen der Werte erfasst und anschließend korrekt auf Xively dargestellt. Die Protokollierung der Aktivitäten des Probanden wird verwendet, um anhand der Überlappung der Graphen die erfassten und visualisierten Daten auszuwerten und zu überprüfen.

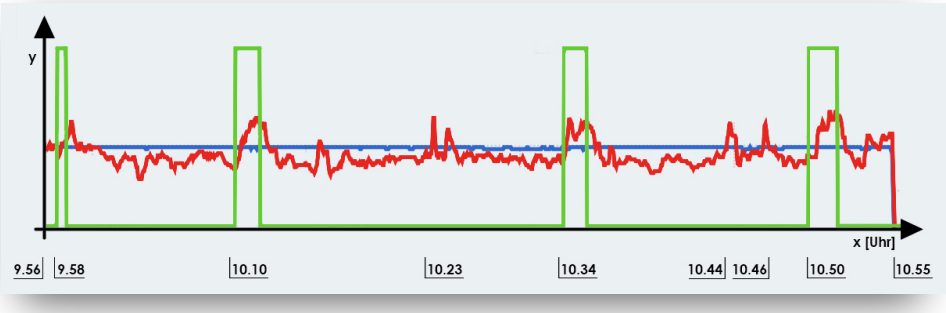


Abbildung 76: Überlappung der Messungen im Systemtest

Der Graph vereint die oben aufgeführten Messungen und zeigt den zeitlichen Verlauf von 9.56 Uhr bis 10.55 Uhr. Der blaue Graph umfasst die Sauerstoffsättigung, der Rote die Pulswerte und der Grüne die Zuckungen, die als Muskelzuckung gedeutet beziehungsweise interpretiert werden.

Das Protokoll des Probanden weist folgende Aktivitäten auf:

1. 9.56 Uhr: Inbetriebnahme des Systems
2. 9:58 Uhr: Zuckung, daraus resultierende Pulswerterhöhung
3. 10.10 Uhr: Zuckung, daraus resultierende Pulswerterhöhung
4. 10.23 Uhr: Gähnen, daraus resultierende Pulswerterhöhung
5. 10.34 Uhr: Zuckung, daraus resultierende Pulswerterhöhung
6. 10.36 Uhr: Gähnen, daraus resultierende Pulswerterhöhung
7. 10.44 Uhr: Gähnen, daraus resultierende Pulswerterhöhung
8. 10.46 Uhr: Gähnen, daraus resultierende Pulswerterhöhung
9. 10.50 Uhr: Zuckung, daraus resultierende Pulswerterhöhung
10. 10.55 Uhr: Ausschalten des Systems

Durch den Vergleich von Protokoll und Graph aus Abbildung 76, wird durch Gähnen wie auch Zuckungen der Puls erhöht. Die Zuckungen hingegen treten nur an den gewollten Positionen auf, sodass keine normalen Bewegungen erfasst wurden. Die Sauerstoffsättigung lässt sich allerdings nicht beeinflussen und so liegt deren Wert konstant zwischen 98 und 99 Prozent. Auch bei der Pulsmessung treten keinerlei Ausreißer auf, sodass die Aussage getroffen werden kann, dass das System bei einer Laufzeit von einer Stunde stabil läuft und keine Fehler aufweist.

5.5.2 Phase 2: Fehlfunktionstest

In diesem Test wird das System auf Fehlfunktionen überprüft, die durch einen Defekt von Hardwarekomponenten oder Ausfallen von einem der beiden Systeme hervorgerufen werden. Auch hierbei wird die Testlaufzeit auf eine Stunde begrenzt, in der von außen die verschiedenen Szenarien am System ausgeführt werden beziehungsweise die Komponenten manipuliert werden. Zum Systemstart wird der Arduino-Produzent mit einem 9V Netzteil mit Strom versorgt und der Client ist dauerhaft an einer festen, konstanten Stromquelle angeschlossen.

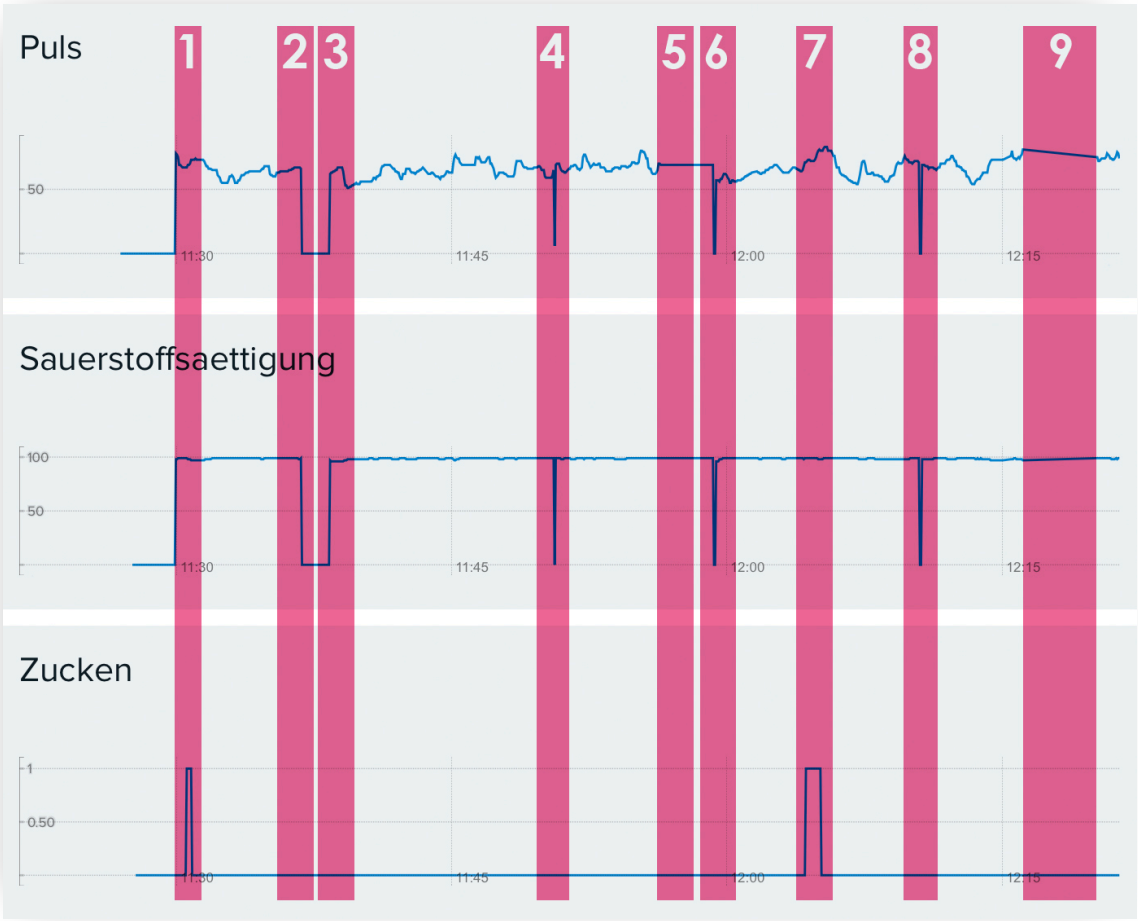


Abbildung 77: Fehlfunktionstest Xively

1. **Sensortest**
   Sensoren werden zu Beginn auf Funktionalität getestet. Alle drei Sensorwerte entsprechen den Anforderungen und arbeiten somit korrekt.
2. **Puls-Oxigen-Sensor Kontaktverlust**
   Der Puls-Oxigen-Sensor wird vom Finger abgenommen und die Werte fallen auf den Wert »0«.Dies entspricht den Anforderungen, da hierdurch visualisiert wird, dass kein Kontakt mehr zum Patienten besteht.
3. **Puls-Oxigen-Sensor Kontakt wiederherstellen**
   Der Sensor wird wieder am Finger angebracht und liefert innerhalb von drei Sekunden wieder die richtigen Werte ohne einen fehlerhaften Wert zu übermitteln.
4. **Wackelkontakt am Sensor verursachen**
   Aufgrund der prototypischen Implementierung entsteht bei zu intensiver Beanspruchung des Sensoranschlusses ein Kontaktverlust, wodurch wiederum der Wert »0« an den Client übermittelt wird. Auch dies hat seine Korrektheit, da aufgrund dessen auf eine Fehlfunktion hingewiesen wird und keine fehlerhaften Werte übermittelt werden.
5. **Arduino-Produzent außer Reichweite des Empfängers**
   Die übermittelten Werte des Arduino-Produzenten können vom Client nicht mehr empfangen werden, daher werden die letzten Werte, welche noch in den Variablen stehen, weiterhin an den Server übermittelt. Auch dies führt nicht zum Zusammenbrechen des Systems, lediglich werden die Werte aufrechterhalten bis der Produzent wieder in Reichweite ist.
6. **Arduino-Client umstellen auf Batterieversorgung**
   Durch das Unterbrechen der Stromversorgung wird der Programmcode bei Wiederanschließen neu gestartet und der Puls-Oxigen-Sensor neu initialisiert. In dieser Zeit übermittelt der Produzent für die Sensoren den Wert »0«.
7. **Sensortest**
   Es wird ein weiterer Sensortest durchgeführt, um die Funktionalität der Sensoren zu überprüfen. Diese arbeiten noch immer korrekt.
8. **Reset des Puls-Oxigen-Sensors**
   Der Puls-Oxigen-Sensor wird neu gestartet, auch hierbei wird bei der Initialisierung der Wert »0« übermittelt.
9. **Arduino-Client kein WLAN-Signal mehr**
   Entstehen Probleme bei der Verbindung über den Router oder der Client ist kurzzeitig ohne Strom, so empfängt Xively keine Daten mehr und wartet bis der nächste Wert wieder übermittelt wird. Ist der Arduino-Client wieder mit Strom versorgt oder das WiFi-Modul konnte wieder eine Verbindung zum Server herstellen, so wird der nächste Wert, der übermittelt wird von Xively in die Datenbank geschrieben und visualisiert, indem der Letzte mit dem aktuell Empfangenen verbunden wird. Daher entsteht im Graph eine lineare Steigung zwischen den beiden Punkten beziehungsweise Werten.

Ergebnis

Das System bricht unter keinem der oben genannten Fehlfunktionen ein, sondern reaktiviert sich und nimmt die Arbeit wieder auf. Unabhängig davon ob eines der Systeme nicht mehr mit Strom versorgt wird, ein Sensor ausfällt oder es zu Kontaktschwierigkeiten an den Modulen kommt. Demnach arbeiten die Systeme autonom und sind unabhängig von der Funktionalität des jeweils anderen.

5.5.3 Überprüfen der ermittelten Anforderungen

In diesem Kapitel wird überprüft, ob die erhobenen Anforderungen an den Prototypen aus der Anforderungsanalyse vom entwickelten System erfüllt werden.

Eingabe/Erfassung

Daten und Informationen über Vitalfunktionen und Bewegungsabläufe des Patienten müssen mittels Sensoren erfasst werden.

- ✓ Vitalfunktionen und Bewegungsabläufe werden vom Puls-Oxigen-Sensor und Accelerometer erfasst.

Verarbeitung

Verwendung eines Niedrigenergie-Standards für die Kommunikation im Home-Area-Netzwerk. Zudem wird ein Arduino-Client als IEEE 802.15.4 zu TCP/IP Gateway benötigt. Dieser ist auch für die Verarbeitung der übermittelten Daten vom Arduino-Produzent zuständig.

- ✓ Verwendung der XBee S1-Module, welche den IEEE 802.15.4-Standard verwenden.
- ✓ Arduino-Client mit XBee-Modul und WiFi-Shield als Gateway eingesetzt.

Ausgabe

Verwendung eines PaaS-Services für die Visualisierung der Daten.

- ✓ Die Daten werden mittels der Xively-JSON-API verarbeitet und auf der Xively-Webseite visualisiert.

Speicherung

Die vom Arduino-Client übermittelten Werte sollen lediglich auf dem Server des Visualisierungsdienstes gespeichert werden. Die Konfigurationsparameter für den Verbindungsaufbau zwischen Arduino-Client und dem Server sollen in den Programmcode geschrieben werden.

- ✓ Die übermittelten Werte werden in der Xively-Datenbank abgelegt und sind über einen langen Zeitraum verfügbar.
- ✓ Die Konfigurationsparameter wie die Netzwerk-SSID und das Passwort werden in Variablen zu Beginn des Programmcodes auf den Arduino-Client geschrieben.

Technische Anforderungen

Nach der Programmierung und Fertigstellung des Prototypen soll das System innerhalb des konfigurierten Netzwerkes funktionsfähig sein, unabhängig davon, ob das System mehrmalig neu gestartet wird.

- ✓ Beide Systeme arbeiten getrennt voneinander. Ist der Arduino-Client für ein Netzwerk konfiguriert, so sind die Konfigurationsparameter persistent beschrieben

und das Board kann beliebig oft neu gestartet werden und bleibt weiterhin einsatzfähig.

Ergonomie der Daten

Die Daten müssen in ein für die API des PaaS-Services lesbares Format gebracht werden.

- ✓ Durch die Implementierung der Xively-Bibliothek wurden im Programmcode die Daten entsprechend den Anforderungen der Xively-JSON-API formatiert.

Leistung

Die Arduino-Boards müssen über genügend Speicherkapazität verfügen, um den Programmcode zu speichern.

- ✓ Beide Systeme wurden mit dem Arduino-UNO umgesetzt und haben nicht einmal die Hälfte des zur Verfügung stehenden Speicherplatzes in Anspruch genommen.

Im Heimnetzwerk muss über den ZigBee- oder IEEE 802.15.4-Standard zwischen den beiden Systemen kommuniziert werden.

- ✓ Es wurden die XB S1-Module eingesetzt, welche den IEEE 802.15.4-Standard verwenden.

Messgenauigkeit

Die Sensoren dürfen nur eine gewisse maximale Toleranz aufweisen. Für die prototypische Implementierung ist es nicht von hoher Relevanz, allerdings galt es zu überprüfen, was möglich ist.

- ✓ Dies ist nur bei der Messung von Puls und Sauerstoffsättigung wichtig. Hierbei wurde ein Sensor aus der Medizin eingesetzt, der den Anforderungen entspricht und über eine Zulassung als medizinisches Produkt verfügt.

Ergebnis

Es wurden alle in der Anforderungsanalyse erhobenen Anforderungen an den Prototyp erfüllt und darüberhinaus auch eine gewisse Stabilität des Systems gewährleistet. Durch den großen Systemtest wie auch die Überprüfung ob das System die Anforderungen erfüllt, wurde eine erfolgreiche Implementierung von Arduino in eine Monitor-Anwendung bewiesen und realisiert.

6 FAZIT

Das Fazit fasst zum einen die in dieser Thesis behandelten Themenbereiche und Aspekte zusammen und bewertet zum anderen die Integration des Arduinos in vernetzte Umgebungen wie auch der prototypischen Implementierung in eine Monitoring-Anwendung.

6.1 Zusammenfassung

Arduino ist weitaus mehr als eine Leiterplatine mit verschiedenen Hardwarekomponenten, welche behilflich sind, die physische Welt zu erfassen und eine Verbindung zur virtuellen Welt herzustellen. Arduino besteht aus Communities, Plattformen und engagierten Entwicklern, die diese Open-Source-Plattform stetig weiterentwickeln.

Durch die bereitgestellte Entwicklungsumgebung und der eigenen Programmiersprache können sogar Programmier-Laien mit diesen Boards umgehen und erste Projekte realisieren. Darüberhinaus existieren viele Klone auf dem Markt, welche eine preiswerte Integration eines Mikrocontrollers in ein Projekt ermöglichen und dennoch kompatibel zu Arduino sind und die Entwicklungsumgebung mitsamt den Bibliotheken nutzen lassen.

Die Nachfrage nach vernetzten Lösungen zum Beispiel in der Hausautomation wächst unaufhaltsam und ist präsenter denn je. Für Hobbybastler und Entwickler bietet auch Arduino hierzu die geeigneten Hardwareerweiterungen an, um den Arduino in eine vernetzte Umgebung zu integrieren. Hierbei wird das Board nicht nur in Verbindung zur Client/Server-Kommunikation über das Internet verwendet, sondern auch im Funkbereich in Heimnetzwerken. Aufgrund der großen Nachfrage bezüglich Lösungen im Phyiscal-Computing-Bereich, können schon jetzt viele Standards in den genannten Bereichen eingesetzt werden, indem ein entsprechendes Erweiterungs-Shield für Arduino bezogen wird. Durch Bibliotheken ist die Einbindung mit nur geringem Aufwand verbunden und lässt somit experimentell in kurzer Zeit eine Idee in eine prototypische Implementierung umsetzen.

Durch verschiedene Sensoren und Aktuatoren kann die Umgebung erfasst werden und zum Beispiel mechanische Komponenten gesteuert werden. In dieser Thesis wurde untersucht, inwiefern ein Arduino-Board als Monitoring-Anwendung im Gesundheitswesen eingesetzt werden kann. Der Anwendungsfall, der dabei durch den Autor festgelegt wurde, befasst sich mit unter Epilepsie leidenden Kindern. Der Prototyp soll hierbei über Sensoren den Schlaf dieser überwachen und die gewonnenen Daten an einen Server übermitteln. Die Implementierung des Boards in eine derartige Anwendung umschließt auch die Integration des Controllers in vernetzte Umgebungen, da für den Heimbereich die über Sensoren erfassten Daten, zum Ermitteln der Vitalfunktionen und Bewegungen, über ein Funknetzwerk übertragen werden und anschließend eine Client/Server-Kommunikation zwischen Arduino und PaaS-Server eine Verbindung über das weltweite Netzwerk abdeckt. Insofern wurden diesbezüglich bei der prototypischen Implementierung zwei vernetzte Umgebungen abgedeckt.

Im Zusammenhang mit der Recherche und Implementierung eines Prototypen wird zudem das Potenzial von Arduino aufgezeigt, welches sich in der einfachen Handhabung, guten Konfigurationsmöglichkeiten und umfangreichen Hardwareerweiterungen beweist.

6.2 Bewertung

Die Anforderungserhebung war auf ein medizinisches Produkt ausgelegt, welches Arduino nicht erfüllen kann. Allerdings liegt die Stärke dieser Boards in der Prototypentwicklung, in der viel ausprobiert und getestet werden muss. Die Anforderungsanalyse hat daher die Bereiche, welche es prototypisch zu erfüllen galt, eingegrenzt und eine Anwendung auf diesem Gebiet realisierbar gemacht. Die meisten Anforderungen auf diesem Gebiet können mit einem Arduino somit realisiert werden, allerdings nicht unter dem Gesichtspunkt, dass das Board in seiner ursprünglichen Form einsetzbar beziehungsweise marktreif wäre. Diesbezüglich sind auf dem Markt spezielle Boards für den E-Health-Bereich erhältlich, wie das »Waspmote« von Libelium, welches die Spezifikationen über ein medizinisches Produkt erfüllt. Arduino wird allerdings viel von Designern und Künstlern für visuelle Effekte eingesetzt, daher galt es eine Anwendung umzusetzen, die ein für den Autor dieser Thesis neues Umfeld für Arduino abdecken soll. Während der Recherche hingegen wurden einige Plattformen, Communities und Hersteller entdeckt, die sich auch mit diesem Bereich auseinandersetzen und versuchen Arduino in medizinische Anwendungsgebiete zu integrieren. Aufgrund dessen ist das Interesse an diesem Bereich größer als vom Autor angenommen.

Es ging nicht nur um eine prototypische Implementierung von Arduino, um aufzuzeigen, dass es möglich ist, Vitalfunktionen zu messen. Die dabei entstandene Idee für ein Produkt erwies sich aufgrund vieler Statements bezüglich dieser Thesis, als dringend notwendig, sodass die Idee für ein derartiges Produkt weiterverfolgt wird, auch wenn das finale Produkt später einmal nicht mit einem Arduino umgesetzt sein wird.

Die Alarmfunktion wurde in der Implementierung nicht umgesetzt, dennoch wäre dies durch ein GSM-Shield von Arduino möglich gewesen, hätte allerdings den Rahmen dieser Thesis überspannt.

Arduino ist vielseitig einsetzbar und überzeugt durch gute Verarbeitung, kostengünstige Komponenten und unzählige Anleitungen im Internet.

6.3 Ausblick

Fast tagtäglich liest man Berichte über neue Geräte aus dem Physical-Computing-Bereich. Computer oder Mikrocontroller integrieren sich immer mehr in den Alltag der Menschheit und übernehmen Aufgaben oder überwachen gewisse Situationen oder Begebenheiten. Die Fraunhofer-Organisation wird in diesem Jahr (2014) höchstwahrscheinlich ein T-Shirt auf den Markt bringen, welches den Pulsschlag des Anwenders misst und auf ein Smartphone übermittelt. Google erwirbt einen Technikhersteller für Thermostate und Rauchmelder für mehrere Milliarden auf. Die Konzerne und Unternehmen rüsten auf und entwickeln Lösungen im Bereich der Hausautomation, im Gesundheitswesen und vielem mehr. Der Trend geht daher immer mehr Richtung vernetzte Umgebungen und Physical-Computing.

Alles muss mit allem verbunden sein und wäre somit von außerhalb über Computer oder Smartphones steuerbar.

Auch im Gesundheitswesen hat Philips schon durch verschiedene Anwendungen versucht, Fuß zu fassen und eine Überwachung der Patienten über das Internet zu realisieren. Allerdings sind die Systeme veraltet und schon zu lange auf dem Markt. Heutzutage sind die Menschen dafür offen und können sich auf derart technischen Lösungen einlassen.

Ob Arduino in diesem Bereich nachziehen wird und wie Libelium ein eigenes Board für diesen Bereich entwickeln wird ist ungewiss. Allerdings ist, wie in dieser Thesis bewiesen wurde, mit den Standard-Boards eine portotypische Implementierung in diesem Gebiet möglich.

LITERATURVERZEICHNIS

[1] "Physical Computing," *Wikipedia*. 25-Okt-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Physical_Computing&oldid=116879148. [Abgerufen am: 30-Oct-2013]

[2] E. Bartmann, *Die elektronische Welt mit Arduino entdecken [mit dem Arduino messen, steuern und spielen ; Elektronik leicht verstehen ; kreativ programmieren lernen ; behandelt Arduino 1.0 ; Buch mit E-Book]*. Köln: O'Reilly, 2012.

[3] D. O'Sullivan, *Physical computing: sensing and controlling the physical world with computers*. Boston: Thomson, 2004.

[4] "Physical Computing 2012 | Physical Computing | 3.Semester | HS 2012," *Physical Computing 2012*, 2012. [Online]. Verfügbar unter: http://blogs.iad.zhdk.ch/physical-computing-hs-12/. [Abgerufen am: 30-Okt-2013]

[5] M. Margolis, *Arduino-Kochbuch [Arduino für Fortgeschrittene ; behandelt Arduino 1.0]*. Köln: O'Reilly, 2012.

[6] D. G. Calin, "Alternative Arduino Boards," *Into Robotics*, 09-Apr-2013. [Online]. Verfügbar unter: http://www.intorobotics.com/alternative-arduino-boards/. [Abgerufen am: 30-Okt-2013]

[7] "Interaction Design Institute Ivrea." [Online]. Verfügbar unter: http://interactionivrea.org/en/about/theinstitute/index.asp. [Abgerufen am: 30-Okt-2013]

[8] T. Brühlmann, *Arduino Praxiseinstieg behandelt Arduino 1.0*. Heidelberg: Verlagsgruppe Hüthig Jehle Rehm, 2012.

[9] M. Odendahl, J. Finn, and A. Wenger, *Arduino - Physical Computing für Bastler, Designer und Geeks*, 2nd ed. Köln[u.a]: O'Reilly, 2010.

[10] "Open Source," *Wikipedia*. 28-Oct-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Open_Source&oldid=123888962. [Abgerufen am: 30-Okt-2013]

[11] "Arduino Introduction," 22-Sep-2010. [Online]. Verfügbar unter: http://www.electronics.dit.ie/staff/tscarff/DT089_Physical_Computing_1/LABS/Arduino_Introduction/Arduino_Introduction.htm. [Abgerufen am: 30-Okt-2013]

[12] "Arduino - Introduction," *Arduino.cc*. [Online]. Verfügbar unter: http://www.arduino.cc/en/Guide/Introduction. [Abgerufen am: 30-Okt-2013]

[13] "Arduino: Mikrocontroller für Einsteiger," *c't*, 2009. [Online]. Verfügbar unter: http://www.heise.de/ct/projekte/Arduino-Mikrocontroller-fuer-Einsteiger-284189.html. [Abgerufen am: 30-Okt-2013]

[14] "AVR - Mikrocontroller.net." [Online]. Verfügbar unter: http://www.mikrocontroller.net/articles/AVR. [Abgerufen am: 30-Oct-2013]

[15] "Boards : Arduino Store - community and electronics," *Arduino.cc*. [Online]. Verfügbar unter: http://store.arduino.cc/eu/index.php?main_page=index&cPath=11_12&sort=20a&page=1. [Abgerufen am: 30-Okt-2013]

[16] "List of Arduino boards and compatible systems," *Wikipedia, the free encyclopedia*. 30-Okt-2013 [Online]. Verfügbar unter: http://en.wikipedia.org/w/index.php?title=List_of_Arduino_boards_and_compatible_systems&oldid=579412575. [Abgerufen am: 30-Okt-2013]

[17] "SainSmart UNO R3 ATmega328-AU Development Board Compatible With

Arduino UNO R3 Specializing in Arduino compatible development boards and modules, oscilloscopes and other electronics.," *Sain Smart*. [Online]. Verfügbar unter: http://www.sainsmart.com/home-page-view/sainsmart-uno-r3-atmega328-au-development-board-compatible-with-arduino-uno-r3.html. [Abgerufen am: 30-Okt-2013]

[18] "Romeo-All in one Controller (Arduino Compatible Atmega 328)," *DF Robot*. [Online]. Verfügbar unter: http://www.dfrobot.com/index.php?route=product/product&product_id=656#.UnEcaZRXmcJ. [Abgerufen am: 30-Okt-2013]

[19] "Lightuino 5.0 - ToastedCircuits - InMojo." [Online]. Verfügbar unter: http://www.inmojo.com/store/toastedcircuits/item/lightuino-5.0/. [Abgerufen am: 30-Okt-2013]

[20] "The JeeLabs Shop - JeeNode (v6)." [Online]. Verfügbar unter: http://jeelabs.com/products/jeenode. [Abgerufen am: 30-Okt-2013]

[21] "Bare Bones Board (BBB) Kit | Modern DeviceModern Device." [Online]. Verfügbar unter: http://moderndevice.com/product/bare-bones-board-bbb-kit/#!prettyPhoto. [Abgerufen am: 30-Okt-2013]

[22] J. Thoma, "Raspberry Pi: Noobs nimmt der Installation den Schrecken - Golem.de," *Golem*, 07-Jun-2013. [Online]. Verfügbar unter: http://www.golem.de/news/raspberry-pi-noobs-nimmt-der-installation-den-schrecken-1306-99690.html. [Abgerufen am: 31-Okt-2013]

[23] "Arduino und Raspberry Pi – Freund oder Feind? - Netzpiloten.de," *Netzpiloten.de – Social Web Explorer*. [Online]. Verfügbar unter: http://www.netzpiloten.de/arduino-und-raspberry-pi-freund-oder-feind/. [Abgerufen am: 30-Okt-2013]

[24] D. Alexandre, "Log real life events in Google Analytics," *Jolicode*, 20-Aug-2013. [Online]. Verfügbar unter: http://jolicode.com/blog/log-real-life-events-in-google-analytics. [Abgerufen am: 30-Okt-2013]

[25] "WiFly-Shield - WiFi/WLAN (SparkFun) - Watterott electronic." [Online]. Verfügbar unter: http://www.watterott.com/de/SparkFun-WiFly-Shield. [Abgerufen am: 04-Nov-2013]

[26] "Arduino - PWM," *Arduino.cc*. [Online]. Verfügbar unter: http://arduino.cc/en/Tutorial/PWM. [Abgerufen am: 30-Okt-2013]

[27] RandomMatrix, "Twitter Mood Light - The World's Mood in a Box," *Instructables.com*. [Online]. Verfügbar unter: http://www.instructables.com/id/Twitter-Mood-Light-The-Worlds-Mood-in-a-Box/. [Abgerufen am: 30-Okt-2013]

[28] "Informationen über den 1-Wire-Bus, Einsatzbereich, Nutzen, Installation." [Online]. Verfügbar unter: http://www.wiregate.de/1-wire-bus. [Abgerufen am: 07-Nov-2013]

[29] "1-Wire," *Wikipedia*. 04-Apr-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=1-Wire&oldid=117063255. [Abgerufen am: 30-Okt-2013]

[30] R. Wenderlich, "Arduino Tutorial: Temperature Sensor," *Arduino Tutorial: Temperature Sensor*, 25-Jul-2013. [Online]. Verfügbar unter: http://www.raywenderlich.com/38841. [Abgerufen am: 30-Okt-2013]

[31] M. Riley, *Das intelligente Haus - Heimautomation mit Arduino und Android und PC*. Köln: O'Reilly, 2012.

[32] R. Sarafan, "Self-Watering Plant," *Instructables.com*. [Online]. Verfügbar unter: http://www.instructables.com/id/Self-Watering-Plant/. [Abgerufen am: 30-Okt-2013]

[33] D. Wheat, *Arduino internals*. [United States]: Apress, 2011 [Online]. Verfügbar unter: http://dx.doi.org/10.1007/978-1-4302-3883-6. [Abgerufen am: 30-Okt-2013]
[34] J. Blum, *Adventures in arduino: tools and techniques for engineering wizardy*, 1st edition. Indianapolis, IN: John Wiley & Sons, 2013.
[35] "ArduinoUno_R3_Front.jpg (1800×1244)," *Arduino.cc*. [Online]. Verfügbar unter: http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg. [Abgerufen am: 30-Okt-2013]
[36] S. Monk, *30 Arduino Selbstbau-Projekte: [die coolsten Bauanleitungen für Arduino]*. Haar bei München: Franzis, 2012.
[37] "Daten im EEPROM speichern, auslesen und Variablen zuweisen," *Fluuux - TechBlog*, 10-Apr-2013. [Online]. Verfügbar unter: http://fluuux.de/2013/04/daten-im-eeprom-speichern-auslesen-und-variablen-zuweisen/. [Abgerufen am: 08-Nov-2013]
[38] "ATMEGA328P-PU Atmel | Mouser," *Mouser Electronics*. [Online]. Verfügbar unter: http://de.mouser.com/Search/ProductDetail.aspx?R=ATMEGA328P-PUvirtualkey55650000virtualkey556-ATMEGA328P-PU. [Abgerufen am: 30-Okt-2013]
[39] "Arduino - ArduinoBoardUno," *Arduino.cc*. [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoBoardUno. [Abgerufen am: 30-Oct-2013]
[40] "AVR TWI - Mikrocontroller.net." [Online]. Verfügbar unter: http://www.mikrocontroller.net/articles/AVR_TWI. [Abgerufen am: 30-Oct-2013]
[41] "Arduino - AnalogRead," *Arduino.cc*. [Online]. Verfügbar unter: http://arduino.cc/de/Reference/AnalogRead. [Abgerufen am: 30-Okt-2013]
[42] "PING))) Ultrasonic Distance Sensor | 28015 | Parallax Inc." [Online]. Verfügbar unter: http://www.parallax.com/product/28015. [Abgerufen am: 30-Okt-2013]
[43] "Arduino - KnockSensor," *Arduino.cc*. [Online]. Verfügbar unter: http://arduino.cc/en/Tutorial/KnockSensor. [Abgerufen am: 30-Okt-2013]
[44] "Arduino Playground - Gyro," *Arduino.cc*. [Online]. Verfügbar unter: http://playground.arduino.cc/Main/Gyro. [Abgerufen am: 30-Okt-2013]
[45] "Baud," *Wikipedia*. 29-Oct-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Baud&oldid=122991207. [Abgerufen am: 06-Nov-2013]
[46] "UART - RN-Wissen." [Online]. Verfügbar unter: http://www.rn-wissen.de/index.php/UART. [Abgerufen am: 06-Nov-2013]
[47] "Universal Asynchronous Receiver Transmitter," *Wikipedia*. 30-Jul-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Universal_Asynchronous_Receiver_Transmitter&oldid=116851542. [Abgerufen am: 06-Nov-2013]
[48] "kreatives-chaos.com » über Mikrocontroller und ihre Schnittstellen." [Online]. Verfügbar unter: http://www.kreatives-chaos.com/artikel/ueber-mikrocontroller-und-ihre-schnittstellen#uart. [Abgerufen am: 06-Nov-2013]
[49] M. Holzer, "Realisierung einer Schnittstelle zur Datenverarbeitung von Arduino Mikrocontrollern über das Internet," Fachhochschule Gießen-Friedberg, 2010 [Online]. Verfügbar unter: http://b00lshit.org/thm/bachelor/thesis.pdf. [Abgerufen am: 10-Sep-2013]
[50] "Arduino - Libraries." [Online]. Verfügbar unter: http://arduino.cc/en/Reference/Libraries. [Abgerufen am: 11-Nov-2013]
[51] "Arduino Shields - Learn.SFE." [Online]. Verfügbar unter: https://learn.sparkfun.com/tutorials/arduino-shields/all. [Abgerufen am: 14-Nov-2013]

[52] "Arduino - ArduinoProtoShield." [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoProtoShield. [Abgerufen am: 14-Nov-2013]
[53] "Arduino - ArduinoBoardFio." [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoBoardFio. [Abgerufen am: 14-Nov-2013]
[54] "Arduino - ArduinoBoardBluetooth." [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoBoardBluetooth. [Abgerufen am: 14-Nov-2013]
[55] "ZigBee Standards Overview." [Online]. Verfügbar unter: http://www.zigbee.org/Standards/Overview.aspx. [Abgerufen am: 14-Nov-2013]
[56] "RFID – Wikipedia," 05-Nov-2013. [Online]. Verfügbar unter: http://de.wikipedia.org/wiki/RFID. [Abgerufen am: 14-Nov-2013]
[57] Torsten, "Funkverbindungen - Arduino auf Distanz," *nextit.de*. [Online]. Verfügbar unter: http://www.nextit.de/2011/07/funkverbindungen-arduino-auf-distanz/. [Abgerufen am: 14-Nov-2013]
[58] "Arduino - ArduinoWirelessShield." [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoWirelessShield. [Abgerufen am: 14-Nov-2013]
[59] "Arduino Wireless SD Shield Tutorial," *Instructables.com*. [Online]. Verfügbar unter: http://www.instructables.com/id/Arduino-Wireless-SD-Shield-Tutorial/?lang=de. [Abgerufen am: 14-Nov-2013]
[60] "XBee® ZB - Digi International." [Online]. Verfügbar unter: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-zb-module. [Abgerufen am: 14-Nov-2013]
[61] "ARDUINO SHD SD - Erweiterung drahtlose Kommunikation, microSD jetzt für nur 23,90 € bei reichelt elektronik," *Elektronik und Technik bei reichelt elektronik günstig bestellen*. [Online]. Verfügbar unter: http://www.reichelt.de/Programmer-Entwicklungstools/ARDUINO-SHD-SD/3/index.html?&ACTION=3&LA=3&ARTICLE=130163&GROUPID=5100. [Abgerufen am: 14-Nov-2013]
[62] "Adafruit PN532 NFC / RFID-Controller-Schild für Arduino + Extras ID: 789 - $ 39.95: Adafruit Industries, Unique & Spaß DIY Elektronik und Kits." [Online]. Verfügbar unter: http://www.adafruit.com/products/789#Description. [Abgerufen am: 14-Nov-2013]
[63] iX, "Berühren statt klicken," *iX*. [Online]. Verfügbar unter: http://www.heise.de/ix/artikel/Beru-hren-statt-klicken-1803728.html. [Abgerufen am: 14-Nov-2013]
[64] "13.56 MHz RFID Transponder." [Online]. Verfügbar unter: http://www.rfid-loesungen.com/HF_RFID_Transponder.htm. [Abgerufen am: 14-Nov-2013]
[65] "Near Field Communication," *Wikipedia*. 14-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Near_Field_Communication&oldid=124445943. [Abgerufen am: 14-Nov-2013]
[66] "Bluetooth 4.0 Low Energy - BLE Shield v2.0," *Seeed Studio Bazaar*. [Online]. Verfügbar unter: http://www.seeedstudio.com/depot/bluetooth-40-low-energy-ble-shield-v20-p-1631.html. [Abgerufen am: 14-Nov-2013]
[67] "Bluetooth Low Energy (BLE) Shields for Arduino 2.0 | Maker Shed," *www.makershed.com*. [Online]. Verfügbar unter: http://www.makershed.com//ProductDetails.asp?ProductCode=MKRBL1. [Abgerufen am: 14-Nov-2013]
[68] "Bluetooth Low Energy," *Wikipedia*. 19-Oct-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Bluetooth_Low_Energy&oldid=122950754. [Abgerufen am: 14-Nov-2013]
[69] "Arduino - ArduinoWiFiShield." [Online]. Verfügbar unter:

http://arduino.cc/de/Main/ArduinoWiFiShield. [Abgerufen am: 15-Nov-2013]
[70] "Shield WiFi.jpg (500×500)." [Online]. Verfügbar unter: http://store.arduino.cc/ww/includes/images/Shield%20WiFi.jpg. [Abgerufen am: 15-Nov-2013]
[71] "Wifly-shield with arduino setup instructions | nickbreen.ca." [Online]. Verfügbar unter: http://nickbreen.ca/blog/wifly-shield-with-arduino-setup-instructions/. [Abgerufen am: 15-Nov-2013]
[72] "arduino wifi shield or wifly rx-xv - Arduino Forum." [Online]. Verfügbar unter: http://forum.arduino.cc/index.php?topic=125242.0. [Abgerufen am: 15-Nov-2013]
[73] "Arduino - ArduinoEthernetShield." [Online]. Verfügbar unter: http://arduino.cc/de/Main/ArduinoEthernetShield. [Abgerufen am: 15-Nov-2013]
[74] "Arduino - ArduinoBoardYun." [Online]. Verfügbar unter: http://arduino.cc/en/Main/ArduinoBoardYun. [Abgerufen am: 16-Nov-2013]
[75] "Arduino Yún | Flikto." [Online]. Verfügbar unter: http://www.flikto.de/arduino-yun.html. [Abgerufen am: 16-Nov-2013]
[76] "EXP TECH." [Online]. Verfügbar unter: http://www.exp-tech.de/Mainboards/Arduino/Arduino-YUN.html. [Abgerufen am: 16-Nov-2013]
[77] "Internet der Dinge," *Wikipedia*. 15-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Internet_der_Dinge&oldid=124486867. [Abgerufen am: 18-Nov-2013]
[78] "Socket (Software) – Wikipedia." [Online]. Verfügbar unter: http://de.wikipedia.org/wiki/Socket_(Software). [Abgerufen am: 18-Nov-2013]
[79] "Client-Server-Modell," *Wikipedia*. 14-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Client-Server-Modell&oldid=124455093. [Abgerufen am: 18-Nov-2013]
[80] C. Schmidt, "Vorlesung Computernetze - Hochschule Offenburg - Fakultät Medien und Informationswesen." .
[81] "Arduino - StreamParseInt." [Online]. Verfügbar unter: http://arduino.cc/de/Reference/StreamParseInt. [Abgerufen am: 21-Nov-2013]
[82] "Yahoo! Weather - Yahoo! Developer Network." [Online]. Verfügbar unter: http://developer.yahoo.com/weather/. [Abgerufen am: 26-Nov-2013]
[83] "Overview: Version 1.1 of the Twitter API | Twitter Developers." [Online]. Verfügbar unter: https://dev.twitter.com/docs/api/1.1/overview#JSON_support_only. [Abgerufen am: 26-Nov-2013]
[84] "Sicherheitsmaßnahmen - Port, NAT, Router & Co - vereinfacht erklärt.," *Forum - CHIP Online*. [Online]. Verfügbar unter: http://forum.chip.de/sicherheitsmassnahmen/port-nat-router-co-vereinfacht-erklaert-1264112.html. [Abgerufen am: 28-Nov-2013]
[85] "Network Address Translation," *Wikipedia*. 25-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Network_Address_Translation&oldid=124485491. [Abgerufen am: 28-Nov-2013]
[86] "Portweiterleitung," *Wikipedia*. 15-Oct-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Portweiterleitung&oldid=120562145. [Abgerufen am: 28-Nov-2013]
[87] "Managed DNS | Outsourced DNS | Anycast DNS." [Online]. Verfügbar unter: http://dyn.com/dns/. [Abgerufen am: 28-Nov-2013]
[88] "Arduino Internet / Web Control with Teleduino." [Online]. Verfügbar unter:

https://www.teleduino.org/. [Abgerufen am: 26-Jan-2014]
[89] "HTTP-Authentifizierung," *Wikipedia*. 08-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=HTTP-Authentifizierung&oldid=122467984. [Abgerufen am: 28-Nov-2013]
[90] S. Hofmann, "Arduino Ethernet Board: Arduino Twitter Library," *Sven Hofmann*. [Online]. Verfügbar unter: http://hofmannsven.com/2013/laboratory/arduino-twitter-library/. [Abgerufen am: 29-Nov-2013]
[91] "Get Twitter library," *Instructables.com*. [Online]. Verfügbar unter: http://www.instructables.com/id/How-to-tweet-from-an-Arduino-using-the-wifi-sheild/step4/Get-Twitter-library/. [Abgerufen am: 29-Nov-2013]
[92] "Apps - ThingSpeak." [Online]. Verfügbar unter: https://www.thingspeak.com/apps. [Abgerufen am: 04-Dez-2013]
[93] "OSI-Modell," *Wikipedia*. 28-Nov-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=OSI-Modell&oldid=124923549. [Abgerufen am: 04-Dez-2013]
[94] "Cloud-Computing," *Wikipedia*. 21-Jan-2014 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Cloud-Computing&oldid=126464085. [Abgerufen am: 26-Jan-2014]
[95] "What is Xively - Xively." [Online]. Verfügbar unter: https://xively.com/whats_xively/. [Abgerufen am: 05-Dez-2013]
[96] "Xively REST API - Xively." [Online]. Verfügbar unter: https://xively.com/dev/docs/api/. [Abgerufen am: 06-Dez-2013]
[97] "Arduino - SD." [Online]. Verfügbar unter: http://arduino.cc/de/Reference/SD. [Abgerufen am: 16-Dez-2013]
[98] "Arduino Playground - Processing." [Online]. Verfügbar unter: http://playground.arduino.cc/interfacing/processing. [Abgerufen am: 16-Dez-2013]
[99] "PulseSensor - Pulse Sensor Amped!" [Online]. Verfügbar unter: http://pulsesensor.myshopify.com/products/pulse-sensor-amped. [Abgerufen am: 16-Dez-2013]
[100] "Klinische Neurophysiologie Göttingen - Epileptische Anfälle und Epilepsien." [Online]. Verfügbar unter: http://www.neurologie.uni-goettingen.de/index.php/epileptische-anfaelle-und-epilepsien.html. [Abgerufen am: 06-Dez-2013]
[101] "Epilepsie: Leben retten mit Überwachung." [Online]. Verfügbar unter: http://www.aerztezeitung.de/medizin/krankheiten/neuro-psychiatrische_krankheiten/epilepsie/article/847129/epilepsie-leben-retten-ueberwachung.html. [Abgerufen am: 06-Dez-2013]
[102] L. J. Meltzer and M. Moore, "Sleep Disruptions in Parents of Children and Adolescents with Chronic Illnesses: Prevalence, Causes, and Consequences," *J. Pediatr. Psychol.*, vol. 33, no. 3, pp. 279–291, Sep. 2007.
[103] "GETEMED Medizin- und Informationstechnik AG: VitaGuard® VG 310." [Online]. Verfügbar unter: http://www.getemed.net/de/monitoring/vitaguardr-vg-310/. [Abgerufen am: 06-Dez-2013]
[104] "EPITECH | Epi-Care 3000." [Online]. Verfügbar unter: http://www.epitech.de/14-0-Epi-Care-3000.html. [Abgerufen am: 06-Dez-2013]
[105] "EPI CARE-Signalgerät: Wie sind eure Erfahrungen? :: REHAkids Das Forum für besondere Kinder :: Das Forum für behinderte Kinder." [Online]. Verfügbar unter: http://www.rehakids.de/phpBB2/ptopic24766.html. [Abgerufen am: 06-Dez-2013]
[106] "Richtlinie 93/42/EWG über Medizinprodukte," *Wikipedia*. 12-Nov-2013 [Online].

Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Richtlinie_93/42/EWG_%C3%BCber_Medizinprodukte&oldid=123136614. [Abgerufen am: 07-Dez-2013]

[107] "ce-richtlinien.eu; CE-Richtlinien; Medizinprodukte 93/42/EWG." [Online]. Verfügbar unter: http://www.ce-richtlinien.eu/richtlinien/MedProd.html. [Abgerufen am: 07-Dez-2013]

[108] "Datei:Zigbee stack.svg," *Wikipedia*. 27-Sep-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Datei:Zigbee_stack.svg&oldid=46244349. [Abgerufen am: 07-Dez-2013]

[109] "IEEE 802.15.4 – Wikipedia." [Online]. Verfügbar unter: http://de.wikipedia.org/wiki/IEEE_802.15.4. [Abgerufen am: 07-Dez-2013]

[110] R. Faludi, *Building wireless sensor networks with ZigBee, XBee, Arduino, and processing*. Farnham: O'Reilly Media, 2010 [Online]. Verfügbar unter: http://proquest.safaribooksonline.com/?fpi=780596807757. [Abgerufen am: 07-Dez-2013]

[111] "Rickshaw: A JavaScript toolkit for creating interactive time-series graphs." [Online]. Verfügbar unter: http://code.shutterstock.com/rickshaw/. [Abgerufen am: 18-Dez-2013]

[112] "Testgetriebene Entwicklung," *Wikipedia*. 27-Dec-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Testgetriebene_Entwicklung&oldid=124942390. [Abgerufen am: 09-Jan-2014]

[113] N. Shibley, "n0m1/MMA8453_n0m1 · GitHub," *an Arduino Library for the Freescale MMA8453Q Accelerometer*, 05-May-2013. [Online]. Verfügbar unter: https://github.com/n0m1/MMA8453_n0m1. [Abgerufen am: 12-Jan-2014]

[114] "Infos und Tipps über Sauerstoffsättigung « Geratherm." [Online]. Verfügbar unter: http://geratherm.de/diagnostik/health-informations/infos-und-tipps-uber-sauerstoffsattigung/. [Abgerufen am: 12-Jan-2014]

[115] "ATmega88.png (1356×704)," *Mikrocontroller.net*. [Online]. Verfügbar unter: http://www.mikrocontroller.net/attachment/137825/ATmega88.png. [Abgerufen am: 30-Okt-2013]

[116] "Simple Mail Transfer Protocol," *Wikipedia*. 04-Oct-2013 [Online]. Verfügbar unter: http://de.wikipedia.org/w/index.php?title=Simple_Mail_Transfer_Protocol&oldid=123141591. [Abgerufen am: 28-Nov-2013]

[117] "Arduino Playground - Shield Pin Usage." [Online]. Verfügbar unter: http://playground.arduino.cc/Main/ShieldPinUsage. [Abgerufen am: 11-Jan-2014]

[118] "eHealth sensor platform schematic." [Online]. Verfügbar unter: http://www.cooking-hacks.com/skin/frontend/default/cooking/images/catalog/documentation/e_health_v2/e-Health_v2.0.pdf

[119] J. Oxer, "Arduino Shield List." [Online]. Verfügbar unter: http://shieldlist.org/. [Abgerufen am: 11-Jan-2014]

QUELLCODEVERZEICHNIS

Quellcode 1: Durch Build-Prozess erzeugte Zwischendatei [5] 45
Quellcode 2: Variablendeklaration 47
Quellcode 3: Konfiguration der seriellen Schnittstelle, Baudrate 9600 48
Quellcode 4: Ethernet initialisieren 51
Quellcode 5: Beispiel Implementierung – Ethernet-Bibliothek [9] 51
Quellcode 6: Ethernet-Bibliothek inkludieren 64
Quellcode 7: Ethernet-Shield feste IP-Adresse festlegen 64
Quellcode 8: DNS und Gateway festlegen 64
Quellcode 9: Ethernet-Initialisierung 64
Quellcode 10: WiFi-Initialisierung 65
Quellcode 11: IP-Adresse asugeben 66
Quellcode 12: Arduino nutzt DNS 67
Quellcode 13: Yahoo-Sketch - Header-Dateien für Bibliotheken inkludieren 68
Quellcode 14: Yahoo-Sketch - MAC- und Server-Adresse festlegen 68
Quellcode 15: Yahoo-Sketch - EthernetClient-Objekt erzeugen 69
Quellcode 16: Yahoo-Sketch - Variable zum Speichern des Ergebnisses 69
Quellcode 17: Yahoo-Sketch - Initialisierung des Ethernet-Shields 69
Quellcode 18: Yahoo-Sketch - loop-Funktion 69
Quellcode 19: Yahoo-Sketch - Verbindungsaufbau zum Yahoo-Server und HTTP-Request 70
Quellcode 20: Yahoo-Sketch - Parsen des HTTP-Response 70
Quellcode 21: Arduino-Server initialisieren 74
Quellcode 22: Arduino-Server - Client-Objekt erzeugen 74
Quellcode 23: Arduino-Server - HTTP-Response-Header 74
Quellcode 24: Arduino-Server - HTML-Formular 75
Quellcode 25: Arduino-Server - Parsen eines HTTP-Requests 76
Quellcode 26: Arduino-Server - HTTP-Post 76
Quellcode 27: Arduino-Mail - Verbindung zum Mail-Server 78
Quellcode 28: Arduino-Mail - Mail-Header und Textkörper 79
Quellcode 29: Arduino-Mail - Mail-Server Authentifizierung 79
Quellcode 30: PHP-Skript Mail-Sender 80
Quellcode 31: Arduino Xively - Xively-Datenstreams 86

Quellcode 32: Datalogging analoger Pins 87
Quellcode 33: Processing-Sketch, serielle Kommunikation [98] 88
Quellcode 34: XBee-Modul Konfiguration 124
Quellcode 35: Arduino XBee-Sender Programmcode 125
Quellcode 36: Beispiel-Sketch dataMode von Noah Shibley 128
Quellcode 37: Beispiel-Sketch Arduino für Processing - Accelerometer 130
Quellcode 38: Accelerometer Arduino-Programmcode für Prototyp 131
Quellcode 39: Beispiel-Sketch Puls-Oxigen-Sensor 135
Quellcode 40: Gesamter Programmcode - Arduino-Produzent 139
Quellcode 41: Arduino-Sketch zum Testen des WiFi-Shields 147
Quellcode 42: Funktion zum Einlesen der Bytes und Parsen 151
Quellcode 43: Programmcode des Arduino-Produzenten, ohne leseSerial()-Funktion 156
Quellcode 44: Arduino Web-Client Ethernet-Sketch XVIII
Quellcode 45: Arduino Web-Client WiFi-Sketch XIX
Quellcode 46: Arduino Web-Server Ethernet-Sketch XX

ANHÄNGE

Anhang A: Pulse-Width-Modulation

PWM ist eine Technik um analoge Werte und Ergebnisse durch digitale Mittel zu erlangen. Digitale Signale können anders als Analoge nur »An« oder »Aus« sein. Bei einem Arduino dahingehend 0V oder 5V. Bei PWM hingegen wird die Dauer der »An«- beziehungsweise 5V-Phasen in Abhängigkeit zu einem bestimmten Zyklus an die Ausgänge geliefert und am Beispiel einer LED, bei einer hohen Frequenz, als für das Auge nicht mehr wahrnehmbares Flackern ausgegeben. Ist die 5V-Phase länger, scheint die LED heller zu leuchten als bei kürzeren »An«-Phasen. Eine Ausgabe über `analogeWrite()` verfügt über einen 256 großen Wertebereich, was wiederum 8 Bits entspricht.

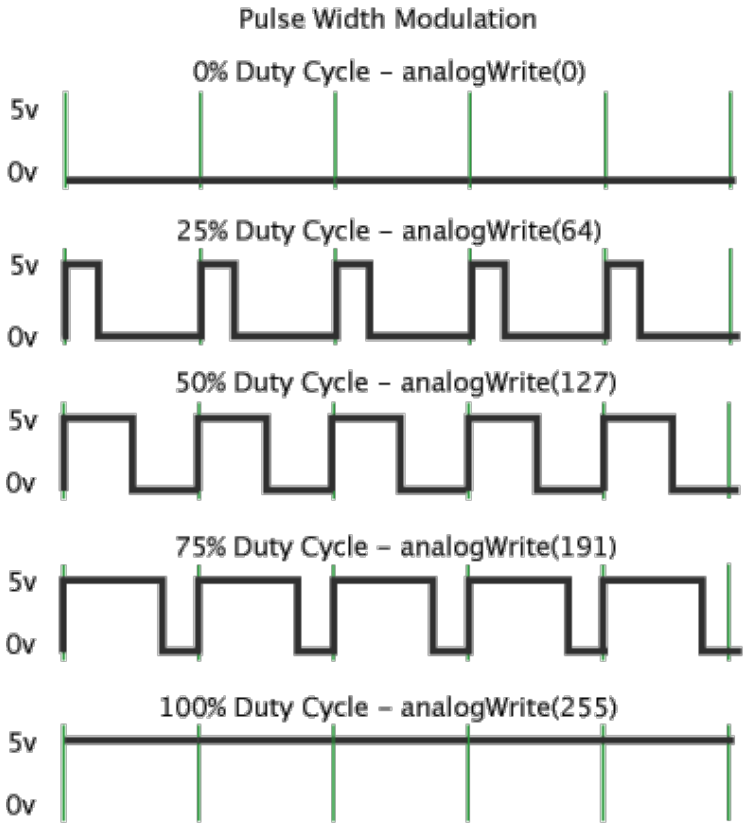


Abbildung 78: Pulse Width Modulation [26]

Anhang B: Standards-Hausautomation

Für die Kommunikation zwischen Geräten werden Feldbussysteme eingesetzt. Die Datenleitung, über welche die Geräte miteinander kommunizieren, wird Feldbus genannt. Hierbei wird von nur einer Leitung ausgegangen worüber die Geräte sich austauschen und Daten übermitteln werden können. Die Geräte wiederum werden Feldgeräte genannt, dabei handelt es sich um Sensoren, Aktuatoren und Raumbediengeräte. Um eine einheitliche und stabile Kommunikation über die Feldbusse zu gewährleisten, sind Protokolle notwendig. In den letzten Jahren haben sich einige Kommunikationsstandards zwar auf dem Markt angesiedelt, aber kaum einer ist weit verbreitet und etabliert. Hierzu gehören zum Beispiel HomePlug, HomePNA, IrDa (Infrared Data Association), Bluetooth, EIB (Europäischer Installationsbus)/KNX (Konex), LonWorks und UPnP (Universal Plug and Play). Des Weiteren gibt es noch ZigBee, ein Industriestandard der oftmals bei XBee-Shields verwendet wird, und BACnet (Building Automation and Control Network).

Bezeichnung	Beschreibung
KNX	Ist der Nachfolger der Feldbusse »EIB«. Der KNX ist ein sehr bekannter Standard in der Home Automation für die Vernetzung verschiedener Geräte und Anlagen über ein Bussystem und unterstützt mehrere Übertragungsmedien. Computer können als IP-Gateway eingerichtet werden und ermöglichen dadurch eine globale Bedienung der KNX-Geräte über das Internet.
LonWorks	Entwickelt von der Firma Echolon, ist die Alternative zum KNX-Standard. Allerdings setzt Echolon mit LonWorks auf eine dezentrale Home Automation. Die Sensoren und Aktuatoren sind mit einem sogenannten Neuron-Chip ausgestattet, wodurch die Kommunikation nicht mehr über einen zentralen Punkt erfolgen muss.
HomePlug	Dies ist ein Anbieter für Steckdosen-Adapter, die eine Power- Line-Communication möglich machen. HomePlug verwendet für die Datenübertragung die Leitungen eines bestehenden Stromnetzes im Haus. Es werden mindestens zwei dieser Adapter benötigt. Einer da- von bei der Steckdose des Senders und der zweite bei der Steck-dose des Empfängers. Je nach Anforderung gibt es unterschiedliche Steckdosen-Adapter. Fu¨r Netzwerkkabel, USB-Kabel und natürlich im Bereich Audio um nur einige Beispiele zu nennen.
HomePNA	Ähnlich wie bei HomePlug gibt es hier auch Adapter, allerdings nicht für das Stromnetz, sondern für die häusliche Telefon- und Koaxialkabel-Leitungen. Das Verkabelungsprinzip ist aber dasselbe, wie bei HomePlug.
UPnP	Damit kann man einfach und schnell Geräte im eigenen Zuhause miteinander vernetzen. Die UPnP Technologie basiert auf einem IP-Netzwerk und jedes IP-fähige Gerät kann darüber angeschlossen und angesteuert werden. UPnP nutzt die bekannten Protokolle

	UDP, TCP/IP und HTTP und lässt deshalb eine Steuerung der vernetzten Geräte über jedes Internetfähige Gerät, wie zum Beispiel das iPhone oder iPad, von außen zu. Für die Kommunikation zwischen den mit UPnP angeschlossenen Geräten wird die Extensible Markup Language (XML) und das Simple Object Access Protocol (SOAP) verwendet. Dadurch, dass UPnP mit den Standard-Protokollen der Netzwerktechnik arbeitet, kann jedes dem Netzwerk neu hinzukommende Gerät selbstständig erkannt werden. Bei einem solchen Vorgang erhält das Gera¨t automatisch eine IP-Adresse, gibt seinen Namen im Netzwerk bekannt und kann des weiteren auf Anfrage auch seine Fähigkeiten offenlegen, sowie eigenständig von anderen Teilnehmern im Netzwerk lernen.
ZigBee	Diese Technologie wurde von der ZigBee Alliance entwickelt. Es handelt sich dabei um ein robustes Funknetz für die Kommunikation zwischen Geräten und Sensoren. Die ZigBee Technologie nutzt den IE- EE 802.15.4 Standard. Aufgrund dieses Standards ist keine Synchronisation für die Kommunikation zwischen den anderen Netzteilnehmern nötig. ZigBee basiert auf der Netzwerk-Topologie Masche.
BACnet	BACnet ist seit dem Jahre 1995 ein aus Amerika stammender Standard für die Gebäudeautomation. Der Datenaustausch erfolgt über Kupferleitungen (Twisted Pair). Ebenso wie KNX und LonWorks findet die Kommunikation über einen Feldbus statt.

Tabelle 5: Kommunikationsstandards der Home Automation

Anhang C: Anschlussbelegung ATMEGA328

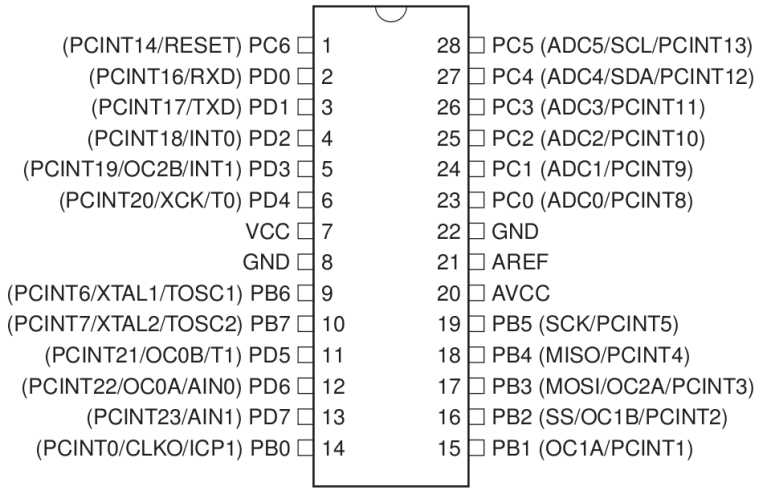


Abbildung 79: ATMEGA328 Belegung [115]

Zuordnung der Pins zu Anschlüssen auf Arduino UNO

ATMEGA-Pin	Pinbeschreibung	Arduino-Board-Funktion
1	PCINT14/RESET (PC6)	Reset
2	PCINT16/RXD (PDo)	D_0 (RX)
3	PCINT17/TXD (PD1)	D_1 (TX)
4	PCINT18/INT_0 (PD2)	D_2
5	PCINT19/OC_2B/INT_1 (PD3)	D_3 (PWM)
6	PCINT20/XCK/T_0 (PD4)	D_4
7	VCC	VCC
8	GND	GND
9	PCINT6/$XTAL_1$/$TOSC_1$ (PB6)	Quarz
10	PCINT7/$XTAL_2$/$TOSC_2$ (PB7)	Quarz
11	PCINT21/OC_0B/T_1 (PD5)	D_5 (PWM)
12	PCINT22/OC_0A/AIN_0 (PD6)	D_6 (PWM)
13	PCINT23/AIN_1 (PD7)	D_7
14	PCINT$_0$/CLKO/ICP_1 (PB0)	D_8
15	OC_1A/PCINT$_1$ (PB1)	D_9 (PWM)
16	SS/OC_1B/PCINT$_2$ (PB2)	D_{10} (PWM)
17	MOSI/OC_2A/PCINT$_3$ (PB3)	D_{11} (PWM)

18	MISO/$PCINT_4$ (PB4)	D_{12}
19	SCK/$PCINT_5$ (PB5)	D_{13}
20	AVCC	VCC
21	AREF	AREF
22	GND	GND
23	ADC_0/$PCINT_8$ (PC0)	A_0
24	ADC_1/$PCINT_9$ (PC1)	A_1
25	ADC_2/$PCINT_{10}$ (PC2)	A_2
26	ADC_3/$PCINT_{11}$ (PC3)	A_3
27	ADC_4/SDA/$PCINT_{12}$ (PC4)	A_4
28	ADC_5/SCL/$PCINT_{13}$ (PC5)	A_5

Tabelle 6: Pins zu Anschlüsse auf Arduino-UNO [8]

Anhang D: Datentypen

Gelistet sind die wichtigsten Datentypen und deren Wertebereiche bei Arduino-Programmen. Weitere Informationen auf *http://arduino.cc/de/Reference/HomePage*.

Datentyp	Wertebereich/Speicherbedarf	Erklärung/Beispiel
int	-32.768 bis 32.767 Speicherbedarf: 2 Bytes	Ganzzahlige Werte für Zähler, Schleifen oder Messwerte ohne Kommastellen Beispiel: `int Abstand=234;`
byte	0 bis 255 Speicherbedarf: 1 Byte	Niedrige Werte (und entsprechend kleiner Speicherbedarf) Beispiel: `byte Helligkeit=126;`
long	-2.147.483.648 bis 2.147.483.647 Speicherbedarf: 4 Bytes	Ganzzahlige Zahlen mit hohen Werten Beispiel: `long durchgaenge=12345;`
float	-3.4028235E+38 bis 3.4028235E+38 Speicherbedarf: 4 Bytes	Werte mit Kommastellen Beispiel: `float TempWert=23.45;`
double	Entspricht dem Datentyp `float`	Beispiel: `Double IRAbstand=432.45;`
char	-128 bis 127 Speicherbedarf: 1 Byte	Wert für die Speicherung eines Zeichens Beispiel: `Char MyBuchstabe='A'`
boolean	TRUE/FALSE Speicherbedarf: 1 Byte	Wert für 1/0, Ja/Nein oder Ein/Aus, also Status EIN oder AUS Beispiel: `Boolean StopStatus=true;`
string	Länge je nach Definition Speicherbedarf: je nach Bedarf	Zeichenketten wie Texte oder mehrere Zeichen Beispiel: `String MyString[ ]="Arduino"`
array	Array mit Größe gemäß Definition Speicherbedarf: je nach Array-Größe	Werte in Variablen mit Tabellenform speichern, wie z.B. Listen von Ports oder Konfigurationswerte Beispiel:

		`int myPorts[ ] = {8, 9, 10, 11};`

Tabelle 7: Wichtige Datentypen für Arduino-Programme [5]

Anhang E: Ethernet Web-Client DNS-Sketch

```
//Quellcode aus der Arduino IDE 1.0.5

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char server[] = "www.google.com

// Statische IP-Adresse falls DHCP fehlschlägt
IPAddress ip(192,168,0,177);

EthernetClient client;

void setup() {

  Serial.begin(9600);

if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    Ethernet.begin(mac, ip);
  }

  delay(1000);
  Serial.println("connecting...");

if (client.connect(server, 80)) {
    Serial.println("connected");
    // Make a HTTP request:
    client.println("GET /search?q=arduino HTTP/1.1");
    client.println("Host: www.google.com");
    client.println("Connection: close");
    client.println();
  }
  else {
    Serial.println("connection failed");
  }
}

void loop()
{
  // if there are incoming bytes Verfügbar unter
  // from the server, read them and print them:
  if (client.Verfügbar unter()) {
    char c = client.read();
    Serial.print(c);
  }

  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();

    // do nothing forevermore:
    while(true);
  }
}
```

Quellcode 44: Arduino Web-Client Ethernet-Sketch

Anhang F: WiFi Web-Client Yahoo-Sketch

```
#include <SPI.h>
#include <WiFi.h>

char ssid[] = "Netzwerkname";
char pass[] = "Passwort";

int status = WL_IDLE_STATUS;

char server[] = "search.yahoo.com";

WiFiClient client;

int result;

void setup() {
  Serial.begin(9600);

  while (status != WL_CONNECTED) {
    status = WiFi.begin(ssid, pass);
    delay(1000);
  }

  if (client.connect(server, 80)) {
    Serial.println("Verbindung erfolgreich");
    client.println("GET /search?p=50+km+in+mi HTTP/1.1");
    client.println();
  }
  else{
    Serial.print("Verbindung fehlgeschlagen");
  }
}

void loop() {

  if(client.connected()){
    if(client.find("Kilometers =")){
       result = client.parseInt();
       Serial.print("50 km sind ");
       Serial.print(result);
       Serial.println(" Meilen.");
    }
  }
  else{
    Serial.println("Verbindung beendet.");
    client.stop();

    while(true);
  }
}
```

Quellcode 45: Arduino Web-Client WiFi-Sketch

Anhang G: Ethernet Web-Server AnalogPin-Sketch

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

EthernetServer server(80);

void setup() {

  Serial.begin(9600);
  Ethernet.begin(mac, ip);
  server.begin();
}

void loop() {

  EthernetClient client = server.Verfügbar unter();
  if (client) {
    Serial.println("new client");
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.Verfügbar unter()) {
        char c = client.read();
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println("Refresh: 5");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          for (int achannel = 0; achannel < 6; achannel ++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        }
        else if (c != '\r')
          currentLineIsBlank = false;
      }
    }
    delay(1);
    client.stop();
    Serial.println("client disonnected");
  }
}
```

Quellcode 46: Arduino Web-Server Ethernet-Sketch

Anhang H: Mail-Header und Textkörper

Client	Server	Erläuterung
`telnet mail.example.com 25`		Client ruft Server
	`220 service ready`	Server meldet sich bereit
`HELO foobar.example.net`		Client nennt seinen Namen
	`250 OK`	Server bestätigt
`MAIL FROM:<sender@example.org>`		Client nennt Absenderadresse
	`250 OK`	Server bestätigt
`RCPT TO:<receiver@example.com>`		Client nennt Empfänger
	`250 OK`	Server bestätigt
`DATA`		Client kündigt Inhalt der E-Mail an
	`354 start mail input`	Server bereit für diesen längeren Vorgang

`FROM: <sender@example.org>`
`TO: <receiver@example.com>`
`Subject: Testmail`
`Date: Thu, 28 Nov 2013`
 	
`Lorem Ipsum dolor sit amet, consectetur adipsad lelit, sed eiusmod`
`.`
		Client sendet Inhalt der E-Mail und markiert das Ende durch eine Zeile, die nur einen Punkt enthält. (Zwischen Header und Textkörper muss eine Leerzeile vorhanden sein)

	`250 OK`	Server bestätigt und übernimmt die Verantwortung für die Nachricht
`QUIT`		Client fordert Verbindungstrennung an
	`221 closing channel`	Server kündigt Trennung an

Tabelle 8: SMTP-Sitzung, Mail-Header und Textkörper [116]

Anhang I: Xively Request-/Response-Header

URL	/api/v2/feeds/2124383220.json
Method	PUT
At	

REQUEST HEADERS

Version	HTTP/1.0
Host	api.xively.com
X-Request-Start	1386280598318046
User-Agent	Xively-Arduino-Lib/1.0
X-Apikey	zmhPPvgWBKSWeNZRi46XhYXl4rEDmFx8XdpMp6bOW2iupd9A
Origin	

REQUEST BODY


```
{
  "version": "1.0.0",
  "datastreams": [
    {
      "id": "sensor_reading",
      "current_value": "903.00"
    }
  ]
}
```


Abbildung 80: Xively Request-Header

Status Code	200

RESPONSE HEADERS

X-Request-Id	246d0a4a64fb0f9f36895102b70dc299ebfb0da4
Cache-Control	max-age=0
Content-Type	application/json; charset=utf-8
Content-Length	0

Abbildung 81: Xively Response-Header

Anhang J: Pinbelegung Arduino-UNO (Produzent)

Module	Spannung		Digitale Pins													Analoge Pins					
	3,3 V	5 V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	0	1	2	3	4	5
Arduino-UNO																						
E-Health-Shield	✓																					
Puls-Oxigen-Sensor									✓	✓	✓	✓	✓	✓	✓	✓						
MMA8452	✓				✓																✓	
XBee-Shield mit Modul		✓	✓	✓																		

Tabelle 9: Pinbelegung Arduino-Produzent [117]–[119]

Anhang K: Pinbelegung Arduino-UNO (Client)

Module	Spannung		Digitale Pins													Analoge Pins					
	3,3V	5 V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	0	1	2	3	4	5
Arduino-UNO																						
WiFi-Shield	✓	✓								✓			✓	✓	✓	✓						
XBee-Shield mit Modul		✓	✓	✓																		

Tabelle 10: Pinbelegung Arduino-Client [69], [119]